LTPDA Toolbox

- LTPDA Toolbox
  - Getting Started with the LTPDA Toolbox
    - What is the LTPDA Toolbox
    - System Requirements
    - Setting-up MATLAB
    - Starting the LTPDA Toolbox
  - Examples
  - Introducing Analysis Objects
    - Creating Analysis Objects
    - Converting existing data into Analysis Objects
    - Saving Analysis Objects
  - Introducing 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
      - Frankin noise-generator
    - Pole/Zero Modelling
      - Creating poles and zeros
      - Building a model
      - Model helper GUI
      - Converting models to IIR filters
  - Standard Math Operations
    - Operators
  - Signal Processing in LTPDA
    - Digital Filtering
      - IIR Filters
      - FIR Filters

    - Spectral Estimation
    - Fitting Algorithms
- Graphical User Interfaces in LTPDA
    - gLTPDA
    - LTPDA Object Browser
    - LTPDA Repository Browser
- 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
    - Setting up an LTPDA Repository -- administrator's guide
- Class descriptions
    - 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
- Functions - By Catagory
- 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. See Setting up an LTPDA Repository for details.

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.

Getting Started with the LTPDA Toolbox System Requirements

©LTP Team

System Requirements

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

- Platforms
- MATLAB and Related Products
- Databases

Platforms

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

MATLAB and Related Products

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

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

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

Component	Version	Comment
MATLAB	7.5 (R2007b)	
Simulink	7.0	Needed for GUI programming

Signal Processing Toolbox	6.8	
Database Toolbox	3.4	Needed to interact with an LTPDA repository

In order to submit and retrieve AOs to/from an LTPDA Repository, your system must have access to an installed MySQL database.

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.
DBDRIVER	'com.mysql.jdbc.Driver'	The name of the database driver for the Database Toolbox. The MySQL driver is shown here.

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

Starting the LTPDA Toolbox is simply a matter of running 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 `startltpgui`.

Setting-up MATLAB Examples

©LTP Team

Examples

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`.

Starting the LTPDA Toolbox Introducing Analysis Objects

©LTP Team

Introducing Analysis Objects

Based on the requirement that all results produced by the LTP Data Analysis software must be easily reproduceable 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

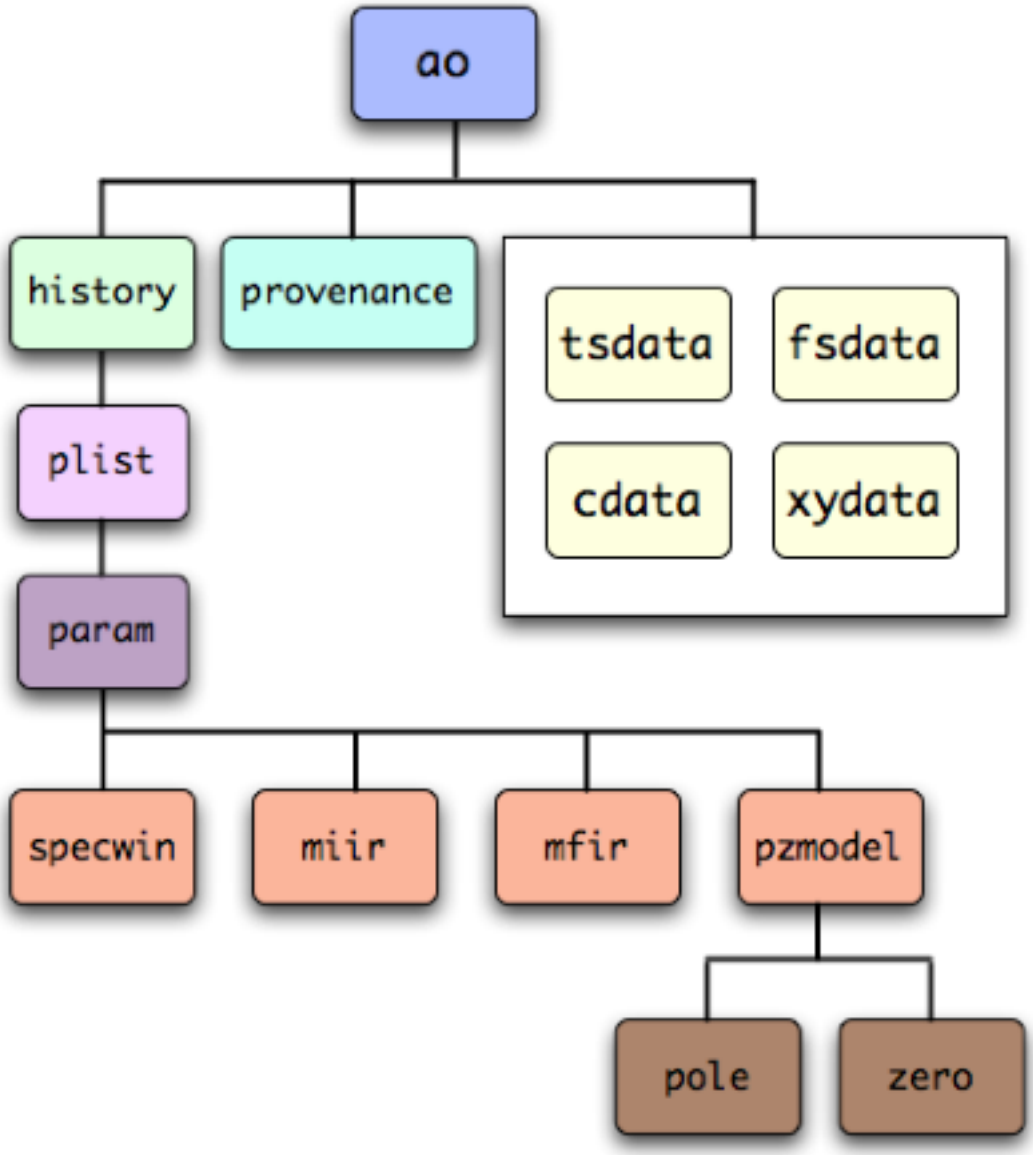
- which raw data was involved (date, channel, time segment, time of retrieval if data can be changed later by new downlinks)
- all operations performed on the data
- the above for all channels of a multi-channel plot

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 (in some cases they are methods of the AO class) but there are also utility functions which do not take AOs as inputs, and do not produce AOs as outputs. Functions in the toolbox are designed to be as simple and elementary as possible.

The functionality of Analysis Objects is built upon a set of MATLAB classes. The classes are depicted in the following Figure.



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

Examples 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_create.html,v 1.8 2008/02/26
09:42:01 hewitson Exp $
    mfile:
-----------------------------
```

As with most constructor calls, an equivalent action can be achieved using an input Parameter List.

```
>> a = ao(plist(param('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')
-- reading provenance
-- reading history
-- reading plist
-- reading param
-- reading history
-- reading plist
-- reading param
-- reading param
-- reading data
-- reading real data samples
* read 10000 Y samples
----------- 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_create.html,v 1.8 2008/02/26
09:42:01 hewitson 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(param('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_create.html,v 1.8 2008/02/26
09:42:01 hewitson 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, param('nsecs', 100));
                 pl = append(pl, param('fs', 10));
                 pl = append(pl, param('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_create.html,v 1.8 2008/02/26
09:42:01 hewitson 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_create.html,v 1.8 2008/02/26
09:42:01 hewitson 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

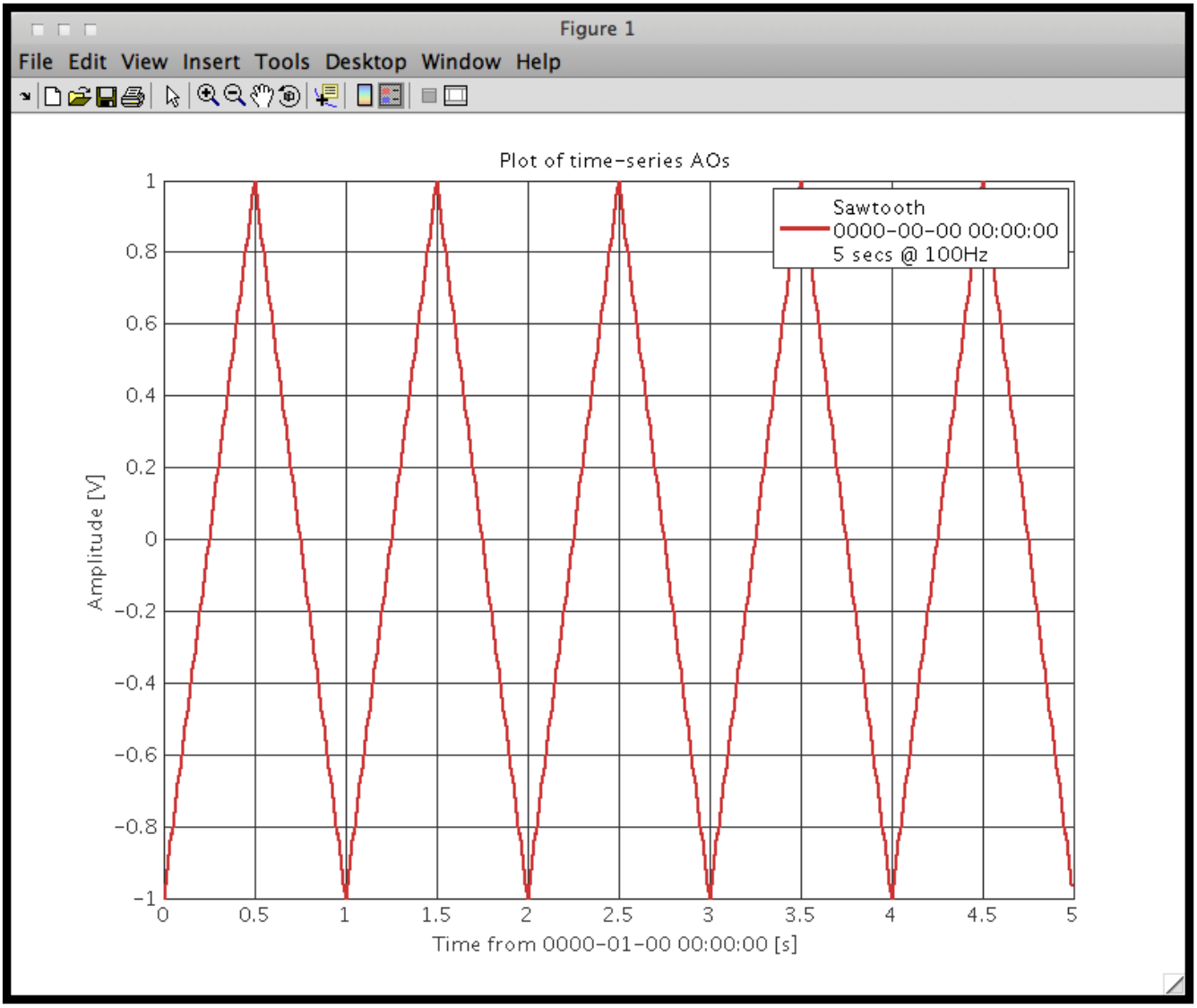
MATLAB contains various functions for creating different waveforms, for example, `square`, `sawtooth`. Some of these functions can be called upon to create Analysis Objects. The following code creates an AO with a sawtooth waveform:

```
                pl = plist();
                pl = append(pl, param('fs', 100));
                pl = append(pl, param('nsecs', 5));
                pl = append(pl, param('waveform', 'Sawtooth'));
                pl = append(pl, param('f', 1));
                pl = append(pl, param('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_create.html,v 1.8 2008/02/26
09:42:01 hewitson Exp $
                      mfile:
                 --------------------------------
```

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

```
                 figure, plot(asaw);
```



Introducing 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 Introducing Parameter Lists

©LTP Team

Introducing 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

©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.

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

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

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.

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

```
>> pl  = append(pl, param('c', 3))  % append a third parameter
n params: 3
---- param 1 ----
key: a
```

```
        val: 1
        -----------------
        ---- param 2 ----
        key: b
        val: 2
        -----------------
        ---- param 3 ----
        key: c
        val: 3
        -----------------
```

Creating Parameters Spectral Windows

©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

©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

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

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

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

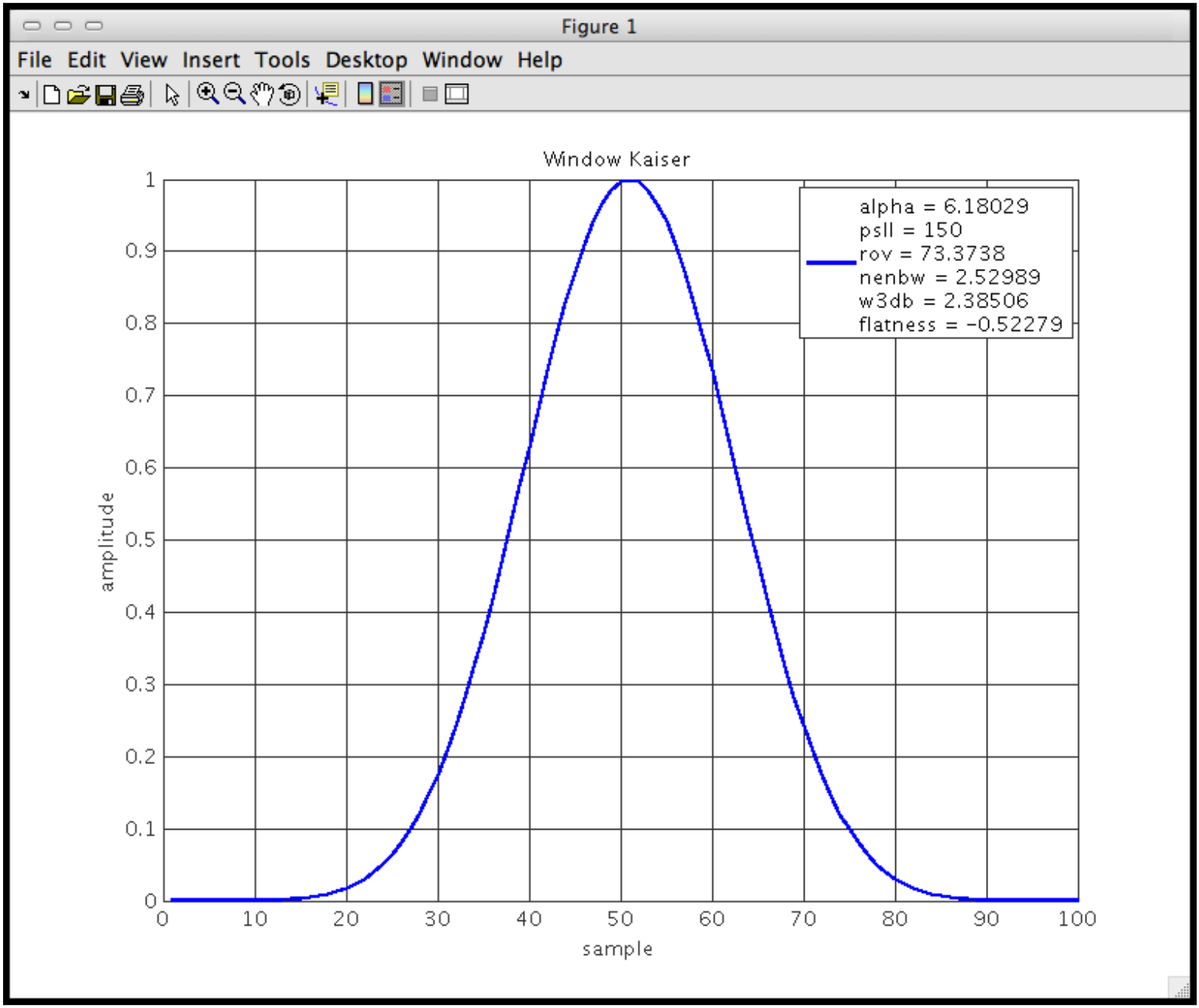
What are LTPDA spectral windows? Visualising spectral windows

©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 MATLAB's `wvtool`. To use `wvtool` with a `specwin` window object, just call:

```
wvtool(w.win);
```

Create spectral windows Using spectral windows

©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

Content needs written...

Simulation/modelling Frankin noise-generator

©LTP Team

Frankin noise-generator

Discussion about Franklin's algorithm.

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

©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 [-7.85398-62.339i]
----------------
```

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();
>> pl  = append(pl, param('name', 'test model'));
>> pl  = append(pl, param('gain', 10));
>> pl  = append(pl, param('poles', [pole(1,2) pole(40)]));
>> pl  = append(pl, param('zeros', [zero(10,3) zero(100)]));
```

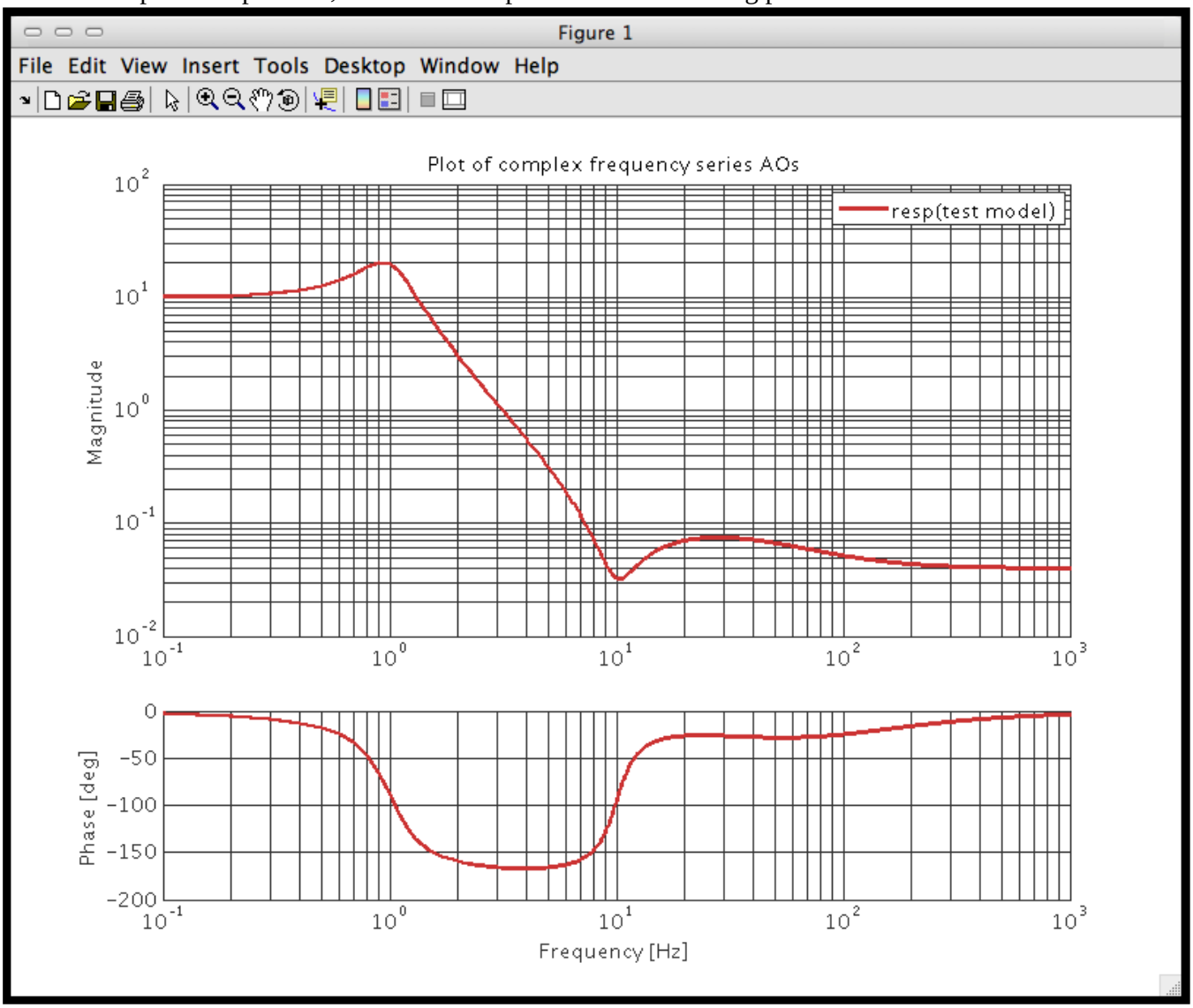
```
>> pzm = pzmodel(pl)
---- pzmodel 1 ----
model: test model
gain: 10
pole 001: pole(1,2)
pole 002: pole(40)
zero 001: zero(10,3)
zero 002: zero(100)
-------------------
```

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();
        >> rpl = append(rpl, param('f1', 0.1));
        >> rpl = append(rpl, param('f2', 1000));
        >> rpl = append(rpl, param('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: pzmodel_model.html,v 1.7 2008/02/26
09:42:01 hewitson Exp $
             mfile:
        ----------------------------
```

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

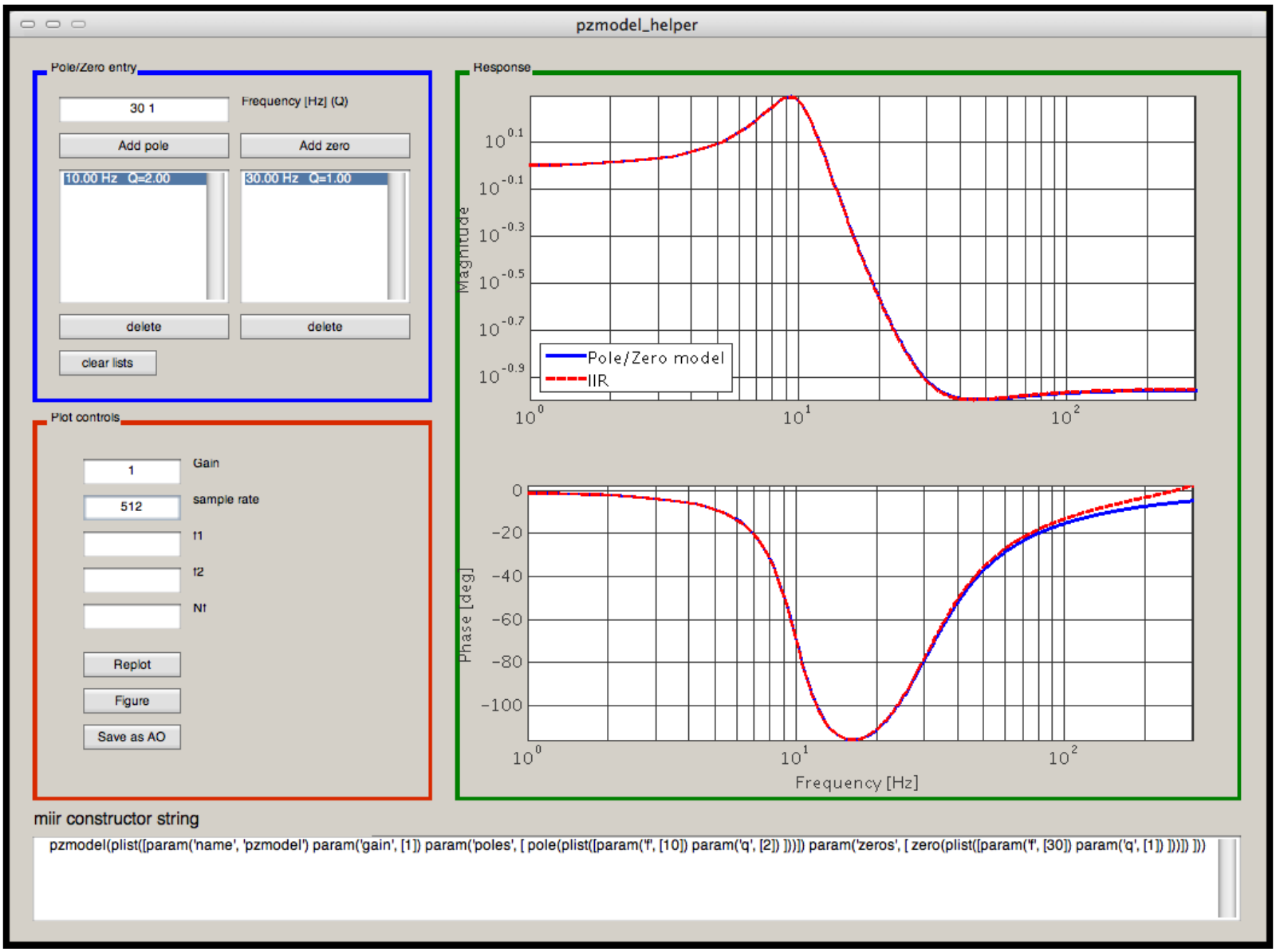
Creating poles and zeros Model helper GUI

©LTP Team

Model helper GUI

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

```
>> pzmodel_helper
```



The GUI also shows the conversion of the model to an IIR filter using a bilinear transform. If no sample rate is set, a value is automatically chosen above the highest zero or pole. You can then explore how accurate the conversion is for different sample rates.

At the bottom of the GUI, a constructor call is shown that produces the model described by the pole and zero lists.

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(param('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([param('pzmodel', pzm) param('fs', 1000)]))
```

For more information of IIR filters in LTPDA, see IIR Filters.

©LTP Team

Standard Math Operations

Converting models to IIR filters Operators

©LTP Team

Operators

Standard Math Operations Signal Processing in LTPDA

©LTP Team

Signal Processing in LTPDA

Content needs written...

Operators Digital Filtering

©LTP Team

Digital Filtering

Content needs written...

Signal Processing in LTPDA IIR Filters

©LTP Team

IIR Filters

Content needs written...

Digital Filtering FIR Filters

©LTP Team

FIR Filters

Content needs written...

IIR Filters Spectral Estimation

©LTP Team

Spectral Estimation

Content needs written...

FIR Filters Fitting Algorithms

©LTP Team

Fitting Algorithms

Content needs written...

Spectral Estimation Graphical User Interfaces in LTPDA

©LTP Team

Graphical User Interfaces in LTPDA

Fitting Algorithms gLTPDA

©LTP Team

gLTPDA

Content needs written...

Graphical User Interfaces in LTPDA LTPDA Object Browser

©LTP Team

LTPDA Object Browser

Content needs written...

©LTP Team

LTPDA Repository Browser

Content needs written...

©LTP Team

Working with an LTPDA Repository

An LTPDA repository serves as a de-centralised storage area for LTPDA objects. Objects can be submitted and retrieved from the repository allowing for easy sharing of objects and for easy recovery of previous analysis results. The following sections discuss the use of an LTPDA repository. The final section, for administrators, discusses the setting up of an LTPDA repository.

What is an LTPDA repository?	An overview of how the repository system works.
Submitting LTPDA objects to a repository	How to submit objects and collections of objects to a repository.
Exploring an LTPDA repository	How to search for objects and explore the contents of a repository.
Retrieving LTPDA objects from a repository	How to retrieve objects and collections from a repository.
Setting up an LTPDA repository -- an administrator's guide	Details for setting up an LTPDA repository.

©LTP Team

What is an LTPDA Repository

Introduction

An LTPDA repository has at its core a MySQL server. A single MySQL server can host multiple LTPDA repositories. A single repository comprises a particular set of database tables.

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`).

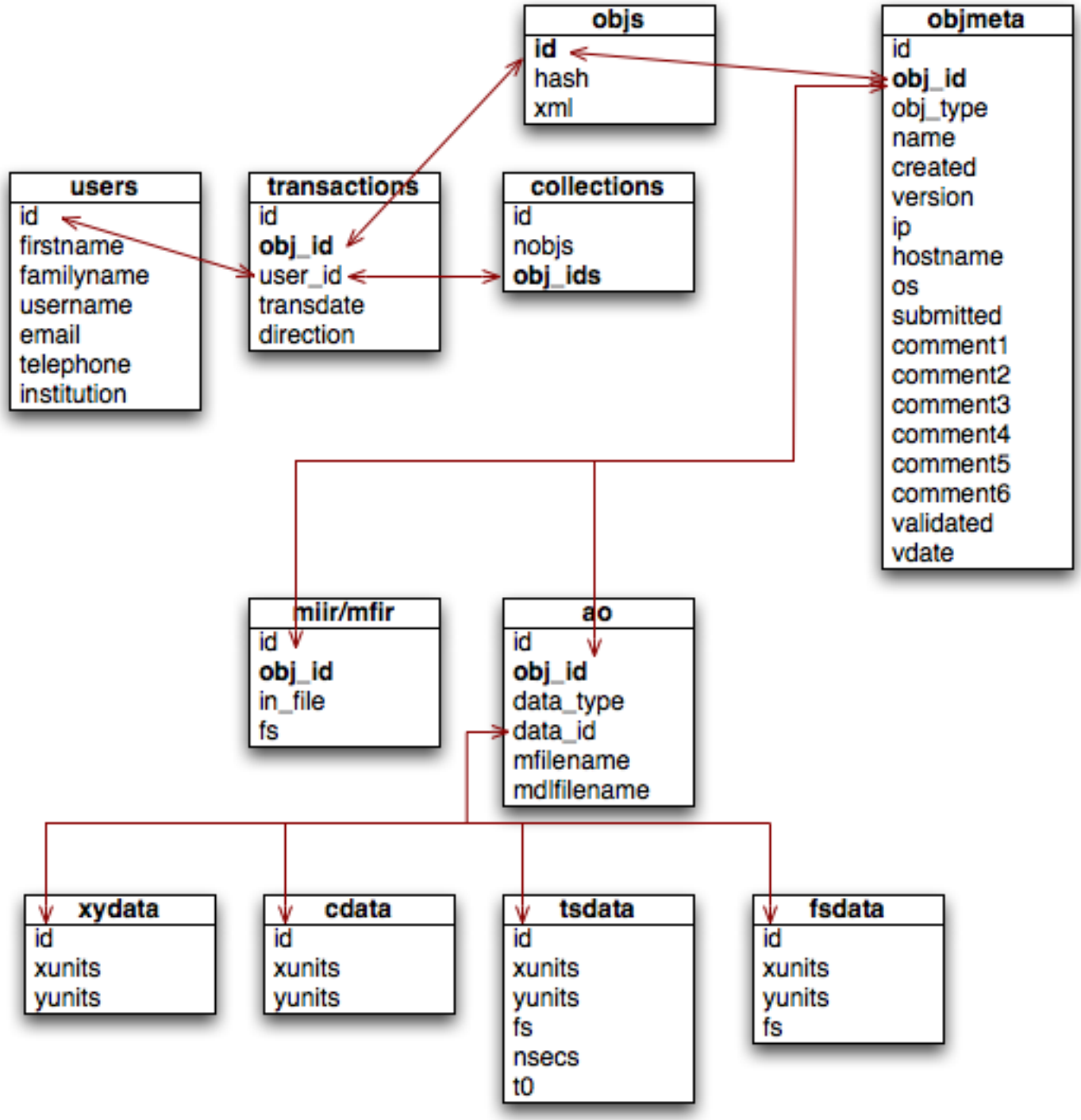
Any standard MySQL client can be used to query and search an LTPDA repository. For example, using a web-client or the standard MySQL command-line interface. In addition, the LTPDA Toolbox provides two ways to search the database: using the command `ltpda_dbquery` or using the LTPDA repository GUI. It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

- An account on the MySQL host server
- Permissions to access the desired database/repository

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. You pass the function a hostname for the repository and a database name to connect to. You will then be prompted for a valid username and password. **Here is an example call:**

```
>> [conn, username] = 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
- time
- timespan
- specwin
- 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. The object to be submitted is checked to be one of the types listed above
3. The `name`, `created`, and `version` fields are read from the object
4. The object is converted to an XML text string
5. An MD5 hash sum is computed for the XML string
6. The XML string and the hash code are inserted in to the `objs` table
7. The automatically assigned ID of the object is retrieved from the `objs` table
8. Various pieces of meta-data (object name, object type, created time, client IP address, etc) are submitted to the `objmeta` table
9. An 'in' entry is made in the `transaction` table recording the user ID and the object ID
10. A entry is then made in the `collections` table, even if this is a single object submission
11. The object IDs and the collection ID are returned to the user

Submitting objects

Objects can be submitted using the command `ltpda_obj_submit`. This command takes two inputs:

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

The information structure should have the following fields:

```
>> sinfo =

                        conn: [1x1 database]
                    username: 'ltpdaadmin'
            experiment_title: 'submit ao'
        experiment_description: 'just a test'
                 reference_ids: ''
           additional_comments: 'none'
            additional_authors: 'no one'
```

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

```
% Connect to a repository
[conn, username] = mysql_connect('localhost', 'ltpda_test');

% Create an AO
p = ao(plist('waveform', 'sine wave', 'f', 1, 'phi', 0, 'nsecs', 3600, 'fs', 10));

% Build an information structure
sinfo.conn                  = conn;
sinfo.username              = username;
sinfo.experiment_title      = 'submit ao';
sinfo.experiment_description = 'this is just a test of the whole thing';
sinfo.reference_ids         = '';
sinfo.additional_comments   = 'none';
sinfo.additional_authors    = 'no one';

% Submit the AO
[ids, cid] = ltpda_obj_submit(p, sinfo);

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

If you run this script (after changing the hostname to a repository where you have permission to connect), you should see the following output:

```
** 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'

*** Submitting objects to repository...
 ** got user id 2 for user: ltpdaadmin
  + submitted object ao with id 1
  + made meta-data entry
  + updated transactions table
 ** made collection entry
*** submission complete.
```

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.

```
% Connect to a repository
[conn, username] = mysql_connect('localhost', 'ltpda_test');

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

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

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

% Close connection
close(conn);

% END
** 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'

*** Submitting objects to repository...
 ** got user id 2 for user: ltpdaadmin
  + submitted object ao with id 16
  + made meta-data entry
  + updated transactions table
  + submitted object specwin with id 17
  + made meta-data entry
  + updated transactions table
  + submitted object plist with id 18
  + made meta-data entry
  + updated transactions table
 ** made collection entry
*** submission complete.
```

If you run the above script (after changing the hostname to a repository where you have permission to connect), you should see the following output:

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.`

Searching particular tables

`it takes the following input arguments:`

`server`	A server hostname
`dbname`	The name of the database to query
`tablename`	The name of the table to search
`query`	The query string

`Examples of usage are:`

```
>> info  = ltpda_dbquery(server, dbname, 'objmeta', 'id>1000 and id<2000');
>> info  = ltpda_dbquery(server, dbname, 'objmeta', 'name like "x12"');
>> info  = ltpda_dbquery(server, dbname, 'users',        'username="aouser"');
>> info  = ltpda_dbquery(server, dbname, 'collections',  'obj_ids="1, 2"');
>> info  = ltpda_dbquery(server, dbname, 'transactions', 'user_id=3');
>> info  = ltpda_dbquery(server, dbname, '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(server, dbname)
```

Executing SQL statements

`For those users who know SQL, the LTPDA toolbox provides a simple mechanism to execute them. The following examples show some typical SQL statements:`

```
                % Get a list of tables
                info  = ltpda_dbquery(server, dbname, 'show tables')

                % Get a table description
                info  = ltpda_dbquery(server, dbname, 'describe objmeta')

                % Get a list of all AO IDs
                info  = ltpda_dbquery(server, dbname, 'select obj_id from objmeta
where obj_type="ao"')

                % Find all objects submitted in a particular time range
                q = ['select obj_id from transactions where '...
                     'transdate>"2007-11-14 00:00:00" and '...
                     'transdate<"2007-11-15 00:00:00"'];
                info  = ltpda_dbquery(server, dbname, q)
```

Submitting LTPDA objects to a repository | Retrieving LTPDA objects from a repository

©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_obj_retrieve` to retrieve objects.

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. The following script shows an example of retrieving a single object:

```
% Connect to a repository
[conn, username] = mysql_connect('130.75.117.67', 'ltpda_test');

% Retrieve the object
q = ltpda_obj_retrieve(conn, 12);

% Close connection
close(conn);
```

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

```
q = ltpda_obj_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] = mysql_connect('130.75.117.67', 'ltpda_test');

% Retrieve the collection
q = ltpda_obj_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 LTPDA Repository GUI

The GUI can be started using the command

```
>> repogui
```

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

Connecting to a repository

The first tab pane is the connection panel. Repo GUI 1

Retrieving LTPDA objects from a repository

Setting up an LTPDA Repository -- administrator's guide

©LTP Team

Setting up an LTPDA Repository -- administrator's guide

Content needs written...

Using the LTPDA Repository GUI Class descriptions

©LTP Team

Class descriptions

Content needs written...

Setting up an LTPDA Repository -- administrator's guide ao Class

©LTP Team

ao Class

Class description for ao class:

Properties	Description
`tag`	Unique tag for this AO
`name`	Name of AO
`data`	Data object associated with this AO
`hist`	History object associated with this AO
`provenance`	Provenance object associated with this AO
`comment`	A human-readable comment describing this AO
`mfile`	full text representation of the m-file that created this AO
`mfilename`	The filename of the m-file that created this AO
`mdlfile`	full text representation of the mdl-file that created this AO
`mdlfilename`	The filename of the mdl-file that created this AO

Class methods

Class descriptions history Class

©LTP Team

history Class

---

ao Class provenance Class

©LTP Team

provenance Class

---

history Class tsdata Class

©LTP Team

tsdata Class

---

provenance Class fsdata Class

©LTP Team

fsdata Class

---

tsdata Class xydata Class

©LTP Team

xydata Class

Content needs written...

fsdata Class cdata Class

©LTP Team

cdata Class

Content needs written...

©LTP Team

plist Class

Content needs written...

cdata Class param Class

©LTP Team

param Class

Content needs written...

©LTP Team

specwin Class

Content needs written...

©LTP Team

miir Class

Content needs written...

specwin Class mfir Class

©LTP Team

mfir Class

Content needs written...

miir Class pzmodel Class

©LTP Team

pzmodel Class

Content needs written...

mfir Class pole Class

©LTP Team

pole Class

Content needs written...

pzmodel Class zero Class

©LTP Team

zero Class

Content needs written...

©LTP Team

Functions - By Catagory

---

zero Class Functions - Alphabetical List

©LTP Team

Functions — Alphabetical List

Functions — by Catagory

Getting Started

©LTP Team

ao

PURPOSE ^

AO analysis object class constructor.

SYNOPSIS ^

function varargout = ao(varargin)

DESCRIPTION ^

```
 AO analysis object class constructor.

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

 DESCRIPTION: AO analysis object class constructor.
              Create an analysis object.

     Properties:
       name       - name of analysis object
       data       - data object (tsdata, fsdata)
       history    - history object
       provenance - creation info, machine, user, etc


     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('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.
                             'columns'      [1 2 1 4]
                                            default = [1 2]
```

```
                                          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)
                              'num_columns'  Maximum colums in the file
                                             default = 10
                              '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'
                              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.
      a = ao(plist)       - creates an analysis object with by reading
                              the file specified by the 'filename'
                              parameter in the plist.
      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

 Parameters for plist constructor:

   'filename' - construct an AO from a filename.
                Example: plist(param('filename', 'a1.xml'))

   'fcn'      - construct an AO from some valid MATLAB function.
                Example: plist(param('fcn', 'randn(100)'))

   'vals'     - construct an AO from a set of values.
                Example: plist(param('vals', [1 2 3]))

   'tsfcn'    - construct an AO from a function of time. You need to also
                specify parameters 'fs' and 'nsecs' for this call.
                Example:
                  plist([param('fs', 10) param('nsecs', 10) param('tsfcn',
                  'sin(2*pi*1.4*t) + 0.1*randn(size(t))')])
```

```
  'fsfcn'    - construct an AO from a function of frequency (f). You also
               need to specify:
                  'f1'    - the initial frequency
                  'f2'    - the final frequency
                  'nf'    - the number of frequency samples
                  'scale' - 'log' or 'lin' frequency spacing
               or provide a frequency vector:
                  'f' - a vector of frequencies on which to evaluate the
                        function

  'win'      - construct an AO from a specwin window function. Tjis
               creates a cdata type AO containing the window values.
               Example: plist(param('win', specwin('Hannning', 100)))

  'waveform' - construct an AO from a waveform description. Here you need
               to specify additional parameters 'fs' and 'nsecs', and also,
               for the following waveform types:
               'sine wave'      - 'f', 'phi'
               '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)

  'Hostname' - construct an AO by retrieving it from an LTPDA repository
               specified by the given hostname. Only those objects which
               are AOs are returned.
               Additional parameters:
               'Database'   - The database name [default: 'ltpda']
               'ID'         - A vector of object IDs.

  'Plist'    - construct from a plist. The value passed should be a plist
               object.

  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 a parameter list object that contains the
default parameter values:

>> pl = ao(ao, 'Params')
```

```
The following call returns a string that contains the routine CVS version:

>> version = ao(ao,'Version')

The following call returns a string that contains the routine category:

>> category = ao(ao,'Category')

See also tsdata, fsdata, xydata, cdata, xyzdata

M Hewitson 30-01-07

%%%
```

- 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.
- imag IMAG overloads the imaginary operator for Analysis objects.
- interp INTERP interpolate the values in the input AO(s) at new values specified by the input parameter list.
- inv INV overloads the inverse function for Analysis Objects.
- join JOIN multiple AOs into a single AO.
- 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 overloads the mean operator for Analysis objects. Compute the mean value.
- median MEDIAN overloads the median operator for Analysis objects. Compute the median value.
- minus MINUS overloads - operator for analysis objects.
- mpower MPOWER implements power operator for analysis objects.
- mrdivide MRDIVIDE overloads / operator for analysis objects.
- mtimes MTIMES overloads * 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 two analysis objects.
- polyfit POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
- power POWER implements power operator for analysis objects.
- rdivide RDIVIDE overloads ./ operator for analysis objects.
- real REAL overloads the real operator for Analysis 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.
- sin SIN overloads the sin operator for Analysis objects. Sine of argument in radians.
- spectrogram SPECTROGRAM computes a spectrogram of the given ao/tsdata.
- split SPLIT split an analysis object into the specified segments.
- sqrt SQRT overloads the sqrt operator for Analysis objects. Square root.
- std STD overloads the std operator for Analysis objects. Compute the standard deviation.
- subsasgn SUBSASGN define index assignment for analysis object properties.
- sum SUM overloads the sum operator for Analysis objects. Compute the sum value.
- svd SVD overloads the determinant function for Analysis objects.
- tan TAN overloads the tan operator for Analysis objects. Tangent of argument in radians.
- times TIMES overloads .* operator for analysis objects.
- transpose TRANSPOSE overloads the .' operator for Analysis Objects.
- upsample UPSAMPLE overloads upsample function for AOs.
- var VAR overloads the var operator for Analysis objects. Compute the variance.

SUBFUNCTIONS ^

- function a = init()
- function aos = aoFromRepository(a, pl, VERSION, ALGONAME)
- function a = aoFromVals(a, vals, VERSION, ALGONAME)
- function a = aoFromWaveform(a, pl, VERSION, ALGONAME)
- function a = aoFromTSfcn(a, pl, VERSION, ALGONAME)
- function a = aoFromFSfcn(a, pl, VERSION, ALGONAME)
- function a = aoFromFcn(a, pl, VERSION, ALGONAME)
- function a = aoFromFilenameAndPlist(init_ao, file_name, pl, VERSION, ALGONAME)

- function a = aoFromSpecWin(a, win, VERSION, ALGONAME)
- function out = getDefaultPlist(varargin)

SOURCE CODE ^

```
function varargout = ao(varargin)
% AO analysis object class constructor.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: AO analysis object class constructor.
%              Create an analysis object.
%
%     Properties:
%       name       - name of analysis object
%       data       - data object (tsdata, fsdata)
%       history    - history object
%       provenance - creation info, machine, user, etc
%
%
%     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('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.
%                             'columns'      [1 2 1 4]
%                                            default = [1 2]
%                                            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)
%                             'num_columns'  Maximum colums in the file
%                                            default = 10
%                             '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'
%                            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.
%       a = ao(plist)       - creates an analysis object with by reading
%                            the file specified by the 'filename'
%                            parameter in the plist.
%       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
%
% Parameters for plist constructor:
%
%   'filename' - construct an AO from a filename.
%                Example: plist(param('filename', 'a1.xml'))
%
%   'fcn'      - construct an AO from some valid MATLAB function.
%                Example: plist(param('fcn', 'randn(100)'))
%
%   'vals'     - construct an AO from a set of values.
%                Example: plist(param('vals', [1 2 3]))
%
%   'tsfcn'    - construct an AO from a function of time. You need to also
%                specify parameters 'fs' and 'nsecs' for this call.
%                Example:
%                  plist([param('fs', 10) param('nsecs', 10) param('tsfcn',
%                  'sin(2*pi*1.4*t) + 0.1*randn(size(t))')])
%
%   'fsfcn'    - construct an AO from a function of frequency (f). You also
%                need to specify:
%                   'f1'    - the initial frequency
%                   'f2'    - the final frequency
%                   'nf'    - the number of frequency samples
%                   'scale' - 'log' or 'lin' frequency spacing
%                or provide a frequency vector:
%                   'f' - a vector of frequencies on which to evaluate the
%                         function
```

```
%
%   'win'      - construct an AO from a specwin window function. Tjis
%                creates a cdata type AO containing the window values.
%                Example: plist(param('win', specwin('Hannning', 100)))
%
%   'waveform' - construct an AO from a waveform description. Here you need
%                to specify additional parameters 'fs' and 'nsecs', and also,
%                for the following waveform types:
%                'sine wave'      - 'f', 'phi'
%                '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)
%
%   'Hostname' - construct an AO by retrieving it from an LTPDA repository
%                specified by the given hostname. Only those objects which
%                are AOs are returned.
%                Additional parameters:
%                'Database'   - The database name [default: 'ltpda']
%                'ID'         - A vector of object IDs.
%
%   'Plist'    - construct from a plist. The value passed should be a plist
%                object.
%
%   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 a parameter list object that contains the
% default parameter values:
%
% >> pl = ao(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = ao(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = ao(ao,'Category')
%
```

```
% See also tsdata, fsdata, xydata, cdata, xyzdata
%
% M Hewitson 30-01-07
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%

%       a = ao('object.mat') - creates a new analysis object by loading the
%                              analysis object from disk .

ALGONAME = mfilename;
VERSION  = '$Id: ao.m,v 1.81 2008/02/25 18:02:45 ingo Exp $';
CATEGORY = 'Constructor';

%%% Check if this is a special call: default parameter list, cvs-version,
if  (nargin == 2 || nargin == 3) &&...
    isa(varargin{1}, 'ao') &&...
    ischar(varargin{2})
  if strcmp(varargin{2}, 'Params')
    if nargin == 2
      varargout{1} = getDefaultPlist();
    else
      varargout{1} = getDefaultPlist(varargin{3});
    end
    return
  elseif strcmp(varargin{2}, 'Version')
    varargout{1} = VERSION;
    return
  elseif strcmp(varargin{2}, 'Category')
    varargout{1} = CATEGORY;
    return
  end
end


% process input arguments
args = [];
k = 0;
pls = [];
for j=1:nargin
  if ~isempty(varargin{j})
    k = k + 1;
    if isa(varargin{j}, 'plist')
      pls = [pls varargin{j}];
    else
      args(k).val = varargin{j};
      args(k).n   = j;
    end
  end
end

% We could have multiple input plist objects. Combine them here.
if isa(pls, 'plist')
```

```
  pl = combine(pls);
else
  pl = [];
end

Nargs = length(args);
if ~isempty(pl)
  args(Nargs+1).val = pl;
  args(Nargs+1).n   = j+1;
  Nargs = length(args);
end

%%%%%%%%%%%%%%%%%%%%   define parameter properties list   %%%%%%%%%%%%%%%%%%%%%

  function a = init()
    a.name        = 'None';
    a.data        = {};
    a.hist        = history(ALGONAME, VERSION);
    a.provenance  = provenance;
    a.description = '';
    a.mfile       = '';
    a.mfilename   = '';
    a.mdlfile     = '';
    a.mdlfilename = '';
    a.plist       = plist();
    a.version     = VERSION;
    a.created     = time;
    a = class(a, 'ao');
  end


%%%%%%%%%%%%%%%%%%%%%%%%%   Create analysis object   %%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%   pl = ao()   %%%%%%%%%%
if Nargs == 0

  varargout{1} = init();

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                              One Parameter                              %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif Nargs == 1              % create ao with data

  %%%%%%%%%%   a1 = ao('foo.mat')   %%%%%%%%%%
  %%%%%%%%%%   a1 = ao('foo.xml')   %%%%%%%%%%
  if ischar(args(1).val)

    a = init();

    filename = args(1).val;
    [path, name, ext, vers] = fileparts(filename);
    switch ext
```

```
      case '.mat'
        a = load(filename);
        a = a.a;
        a.name = strcat(name, ext);
      case '.xml'
        root_node = xmlread(filename);
        a = ltpda_xmlread(root_node, 'ao');
      case {'.txt', '.dat'}
        a = aoFromFilenameAndPlist(a, args(1).val, plist(), VERSION, ALGONAME);
        a.name = strcat(name, ext);
      otherwise
        error('### Unknown file type.');
    end
    varargout{1} = a;

  %%%%%%%%%%   a1 = ao(ao)   %%%%%%%%%%
  elseif isa(args(1).val, 'ao')
    varargout{1} = args(1).val;

  %%%%%%%%%%   a1 = ao(struct)   %%%%%%%%%%
  % create from struct
  elseif isstruct(args(1).val)
    a       = init();
    astruct = args(1).val;
    a.name  = astruct.name;
    data    = astruct.data;

    if isstruct(data)
      if isfield(data, {'t', 'x'})
        data = tsdata(data);
      elseif isfield(data, {'f', 'xx'})
        data = fsdata(data);
      elseif isfield(data, {'vals'})
        data = cdata(data);
      elseif isfield(data, {'x', 'y'})
        data = xydata(data);
      elseif isfield(data, {'x', 'y', 'z'})
        data = xyzdata(data);
      else
        error('### Unknown data type in input structure.');
      end
    end

    a.data = data;
    hist   = astruct.hist;
    if isstruct(hist)
      hist = history(hist);
    end
    a.hist = hist;
    prov = astruct.provenance;
    if isstruct(prov)
      prov = provenance(prov);
```

```
    end

    a.provenance  = prov;
    a.description = astruct.description;
    a.mfile       = astruct.mfile;
    a.mfilename   = astruct.mfilename;
    a.mdlfile     = astruct.mdlfile;
    a.mdlfilename = astruct.mdlfilename;
    a.plist       = astruct.plist;
    a.version     = astruct.version;
    created       = astruct.created;
    if isstruct(created)
      created = time(created);
    end
    a.created     = created;
    varargout{1}  = a;

  %%%%%%%%%%   a1 = ao(1)   %%%%%%%%%%
  % create from constant
  elseif isnumeric(args(1).val)

    a      = init();
    a.name = 'const';
    a.data = cdata(args(1).val);
    a.hist = history(ALGONAME, VERSION, plist(param('vals', args(1).val)));

    varargout{1}  = a;

  %%%%%%%%%%   a1 = ao(plist)   %%%%%%%%%%
  % create data using plist
  elseif isa(args(1).val, 'plist')

    pl       = args(1).val;
    filename = find(pl, 'filename');
    fcn      = find(pl, 'fcn');
    vals     = find(pl, 'vals');
    tsfcn    = find(pl, 'tsfcn');
    fsfcn    = find(pl, 'fsfcn');
    win      = find(pl, 'win');
    waveform = find(pl, 'waveform');
    hostname = find(pl, 'hostname');
    ipl      = find(pl, 'Plist');

    a = init();
    a.plist = pl;

    if ~isempty(filename)
      % do filename constructor
      [path, name, ext, vers] = fileparts(filename);
      switch ext
        case '.mat'
```

```
          a = load(filename);
          varargout{1} = a.a;
        case '.xml'
          root_node = xmlread(filename);
          a = ltpda_xmlread(root_node, 'ao');
          varargout{1} = a;
        case {'.txt', '.dat'}
          ao_out = aoFromFilenameAndPlist(a, filename, pl, VERSION, ALGONAME);
          if nargout == 1
            varargout{1} = ao_out;
          elseif nargout == length(ao_out)
            for ii = 1:length(ao_out)
              varargout{ii} = ao_out(ii);
            end
          else
            warning ('### Too few output variables --> useing only the first.');
            varargout{1} = ao_out;
          end
        otherwise
          error('### Unknown file type.');
      end

    elseif ~isempty(fcn)
      % do function constructor
      varargout{1} = aoFromFcn(a, pl, VERSION, ALGONAME);

    elseif ~isempty(vals)
      % do vals constructor
      varargout{1} = aoFromVals(a, vals, VERSION, ALGONAME);
    elseif ~isempty(tsfcn)
      % do tsfcn constructor
      varargout{1} = aoFromTSfcn(a, pl, VERSION, ALGONAME);
    elseif ~isempty(fsfcn)
      % do fsfcn constructor
      varargout{1} = aoFromFSfcn(a, pl, VERSION, ALGONAME);
    elseif ~isempty(win)
      % do win constructor
      varargout{1} = aoFromSpecWin(a, win, VERSION, ALGONAME);
    elseif ~isempty(waveform)
      % do waveform constructor
      varargout{1} = aoFromWaveform(a, pl, VERSION, ALGONAME);
    elseif ~isempty(hostname)
      % do hostname constructor
      varargout{1} = aoFromRepository(a, pl, VERSION, ALGONAME);
    elseif ~isempty(ipl)
      % do plist constructor
      varargout{1} = ao(ipl);
    else
      error('### Unknown AO constructor method.');
    end

  %%%%%%%%%%   a1 = ao(specwin)   %%%%%%%%%%
```

```
  % create from spectral window
  elseif isa(args(1).val, 'specwin')
    a = init();
    varargout{1} = aoFromSpecWin(a, args(1).val, VERSION, ALGONAME);

  %%%%%%%%%%   a1 = ao(cdata)    %%%%%%%%%%
  %%%%%%%%%%   a1 = ao(fsdata)   %%%%%%%%%%
  %%%%%%%%%%   a1 = ao(tsdata)   %%%%%%%%%%
  %%%%%%%%%%   a1 = ao(xydata)   %%%%%%%%%%
  % create from data object
  else
    a = init();
    data          = args(1).val;
    a.name        = 'None';
    a.data        = data;
    a.hist        = history(ALGONAME, VERSION);

    if ~isa(a.data, 'tsdata') &&...
       ~isa(a.data, 'fsdata') &&...
       ~isa(a.data, 'cdata')  &&...
       ~isa(a.data, 'xydata')  &&...
       ~isa(a.data, 'xyzdata')
      error('### unknown data object.')
    end
    varargout{1} = a;
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                              Two Parameter                               %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

elseif Nargs == 2

  %%%%%%%%%%   Database   %%%%%%%%%%
  if isa(varargin{1}, 'database')
    varargout{1} = retrieve(varargin{1}, varargin{2:end});

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

    a = init();

    ao_out = aoFromFilenameAndPlist(a, args(1).val, args(2).val, VERSION, ALGONAME);

    for ii = numel(ao_out)
      ao_out(ii).plist = args(2).val;
      ao_out(ii).name = args(1).val;
    end

    if nargout == 1
      varargout{1} = ao_out;
```

```
    elseif nargout == length(ao_out)
      for ii = 1:length(ao_out)
        varargout{ii} = ao_out(ii);
      end
    else
      warning ('### Too few output variables --> useing only the first.');
      varargout{1} = ao_out;
    end

  %%%%%%%%%%%   ao(data-object, history-object)   %%%%%%%%%%
  else

    a = init();

    a.data        = args(1).val;
    a.hist        = args(2).val; %history(ALGONAME, VERSION, [], varargin{2});

    if ~isa(a.data, 'tsdata') &&...
        ~isa(a.data, 'fsdata') &&...
        ~isa(a.data, 'cdata')  &&...
        ~isa(a.data, 'xydata')
      error('### unknown data object.')
    end
    if ~isa(a.hist, 'history')
      error('### unknown history object.')
    end

    varargout{1} = a;

  end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                             Nine Parameter                              %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  full constructor - all fields
elseif Nargs == 9

  a = init();
  a.name        = args(1).val;
  a.data        = args(2).val;
  a.hist        = args(3).val;
  a.provenance  = args(4).val;
  a.description = args(5).val;
  a.mfile       = args(6).val;
  a.mfilename   = args(7).val;
  a.mdlfile     = args(8).val;
  a.mdlfilename = args(9).val;
  varargout{1}  = a;

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

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    aoFromRepository
%
% DESCRIPTION: Construct an ao from a repository
%
% CALL:        aos = aoFromRepository(a, pl, VERSION, ALGONAME)
%
% PARAMETER:   a:         Initialised analysis object
%              pl:        Parameter list object
%              VERSION:   cvs version string
%              ALGONAME: The m-file name (use the mfilename command)
%
% HISTORY:     07-05-2007 Hewitson
%              Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function aos = aoFromRepository(a, pl, VERSION, ALGONAME)

  dpl = getDefaultPlist('From Repository');
  pl  = combine(pl, dpl);

  % 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);
  aos  = [];
  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
      a = ltpda_obj_retrieve(conn, id);
      %---- Add history

      % make a new history object
      %  - remove connection object from plist first
      h = history(ALGONAME, VERSION, remove(pl, 'conn'), a.hist);

      % set this to the AO
      a = setnh(a, 'hist', h);

      %---- Add to output array
      aos = [aos a];
    else
      warning('    !skipping ID %d, type %s', id, tt);
    end

  end

  % close connection
  if closeConn
    close(conn);
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    aoFromVals
%
% DESCRIPTION: Construct an ao from a value set
%
% CALL:        a = aoFromVals(a, vals, VERSION, ALGONAME)
%
% PARAMETER:   a:        Initialised analysis object
%              vals:     Constant values
%              VERSION:  cvs version string
%              ALGONAME: The m-file name (use the mfilename command)
%
% HISTORY:     07-05-2007 Hewitson
%              Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function a = aoFromVals(a, vals, VERSION, ALGONAME)

  if ischar(vals)
    try
      vals = eval(vals);
    catch
      error('Could not convert char array to numeric array.');
    end
  end

```

```
  % Make a cdata object
  ts = cdata(plist(param('vals', vals)));
  ts = set(ts, 'name', 'Data');

  % Make an analysis object
  a.name = 'Data';
  a.data = ts;
  a.hist = history(ALGONAME, VERSION, plist(param('vals', vals)));
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    aoFromWaveform
%
% DESCRIPTION: Construct an ao from a waveform
%
% CALL:        a = aoFromWaveform(a, pl, VERSION, ALGONAME)
%
% PARAMETER:   a:        Initialised analysis object
%              pl:       Parameter list object
%              VERSION:  cvs version string
%              ALGONAME: The m-file name (use the mfilename command)
%
% HISTORY:     07-05-2007 Hewitson
%              Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function a = aoFromWaveform(a, pl, VERSION, ALGONAME)

  nsecs  = find(pl, 'nsecs');
  fs     = find(pl, 'fs');
  t0     = find(pl, 't0');
  if isempty(t0)
    t0 = time(0);
  end

  waveform = find(pl, 'waveform');
  switch waveform
    %------------ Sine Wave
    case 'sine wave'
      freq = find(pl, 'f');
      phi  = find(pl, 'phi');
      if isempty(freq)
        freq = 1;
      end
      if isempty(phi)
        phi = 0;
      end
      tsfcn = sprintf('sin(2*pi*%g*t + %g*pi/180)', freq, phi);
      %------------ Noise
    case 'noise'
```

```
      ntype = find(pl, 'type');
      if isempty(ntype)
        ntype = 'Normal';
      end
      switch ntype
        case 'Normal'
          tsfcn = 'randn(size(t))';
        case 'Uniform'
          tsfcn = 'rand(size(t))';
      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'
      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'
      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 'Sawtooth'
      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 = set(ts, 'name', tsfcn);
  ts = set(ts, 'xunits', 's');
  ts = set(ts, 'yunits', 'V');
  ts = set(ts, 't0', t0);

  % Make an analysis object
  a.name = waveform;
  a.data = ts;
  a.hist = history(ALGONAME, VERSION, pl);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    aoFromTSfcn
%
% DESCRIPTION: Construct an ao from a ts-function string
%
% CALL:        a = aoFromTSfcn(a, pl, VERSION, ALGONAME)
%
% PARAMETER:   a:        Initialised analysis object
%              pl:       Parameter list object
%              VERSION:  cvs version string
%              ALGONAME: The m-file name (use the mfilename command)
%
% HISTORY:     07-05-2007 Hewitson
%              Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function a = aoFromTSfcn(a, pl, VERSION, ALGONAME)

  nsecs       = find(pl, 'nsecs');
  fs          = find(pl, 'fs');
  fcn         = find(pl, 'tsfcn');
  randn_state = find(pl, 'rand_state');
  t0          = find(pl, 't0');
  if isempty(t0)
    t0 = time(0);
  end
```

```

  % construct tsdata
  t = linspace(0, nsecs-1/fs, nsecs*fs);

  % make y data

  % 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(pl , 'rand_state', randn_state);
  end

  y = eval([fcn ';']);

  ts = tsdata(t,y);
  ts = set(ts, 'name', fcn);
  ts = set(ts, 'xunits', 's');
  ts = set(ts, 'yunits', 'V');
  ts = set(ts, 't0', t0);

  % Make an analysis object
  a.name = 'TSfcn';
  a.data = ts;
  a.hist = history(ALGONAME, VERSION, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    aoFromFSfcn
%
% DESCRIPTION: Construct an ao from a fs-function string
%
% CALL:        a = aoFromFSfcn(a, pl, VERSION, ALGONAME)
%
% PARAMETER:   a:        Initialised analysis object
%              pl:       Parameter list object
%              VERSION:  cvs version string
%              ALGONAME: The m-file name (use the mfilename command)
%
% HISTORY:     07-05-2007 Hewitson
%              Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function a = aoFromFSfcn(a, pl, VERSION, ALGONAME)

  dpl = getDefaultPlist('Frequency-series Function');
  pl  = combine(pl, dpl);
  f   = find(pl, 'f');
  if isempty(f)
```

```
    disp('--- generating f')
    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 = set(fs, 'name', fcn);
  fs = set(fs, 'xunits', 'Hz');
  fs = set(fs, 'yunits', 'Arb');

  % Make an analysis object
  a.name = 'FSfcn';
  a.data = fs;
  a.hist = history(ALGONAME, VERSION, pl);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    aoFromFcn
%
% DESCRIPTION: Construct an ao from a function string
%
% CALL:        a = aoFromFcn(a, pl, VERSION, ALGONAME)
%
% PARAMETER:   a:        Initialised analysis object
%              pl:       Parameter list object
%              VERSION:  cvs version string
%              ALGONAME: The m-file name (use the mfilename command)
%
% HISTORY:     07-05-2007 Hewitson
%              Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function a = aoFromFcn(a, pl, VERSION, ALGONAME)

```

```
  fcn = find(pl, 'fcn');

  % Make a cdata object
  ts = cdata(plist(param('fcn', fcn)));
  ts = set(ts, 'name', fcn);

  % Make an analysis object
  a.name = fcn;
  a.data = ts;
  a.hist = history(ALGONAME, VERSION, pl);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    aoFromFilenameAndPlist
%
% DESCRIPTION: Construct an ao from filename AND parameter list
%
% CALL:        a = aoFromFilenameAndPlist(init_ao, file_name, pl, VERSION, ALGONAME)
%
% PARAMETER:   init_ao:   Initialised analysis object
%              file_name: File name
%              pl:        Parameter list object
%              VERSION:   cvs version string
%              ALGONAME:  The m-file name (use the mfilename command)
%
% HISTORY:     07-05-2007 Hewitson
%              Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function a = aoFromFilenameAndPlist(init_ao, file_name, pl, VERSION, ALGONAME)

  a = [];

  [pathstr, f_name, ext] = fileparts(file_name);

  %%%%%%%%%%   Define default parameter list   %%%%%%%%%%
  pl_default = plist('columns',      [1 2],    ...
                     'num_columns',   10,      ...
                     'type',         'tsdata', ...
                     'use_fs',       '',       ...
                     'comment_char', '%',      ...
                     'xunits',       'sec',    ...
                     'yunits',       'V');
  pl = combine(pl, pl_default);

  pl = pset(pl, param('filename', [f_name ext]));
  pl = pset(pl, param('filepath', pathstr));

  data_type    = find (pl, 'type');
```

```
  columns      = find (pl, 'columns');
  num_columns  = find (pl, 'num_columns');
  comment_char = find (pl, 'comment_char');
  use_fs       = find (pl, 'use_fs');
  xunits       = find (pl, 'xunits');
  yunits       = find (pl, 'yunits');
  t0           = find (pl, 't0');

  %%%%%%%%%%   read file   %%%%%%%%%%
  [fid,msg] = fopen (file_name, 'r');
  if (fid < 0)
    error ('### can not open file: %s \n\n### error msg:',file_name, msg);
  end

  %%%%%%%%%% create scan format: '%f %f %f %f %f'   %%%%%%%%%%
  scan_format = [];
  for ii=1:num_columns
    scan_format = [scan_format '%f '];
  end
  scan_format = scan_format(1:end-1); % remove the last whitespace

  %%%%%%%%%%   scan the file   %%%%%%%%%%
  raw_data = textscan( fid, scan_format,             ...
                       'commentStyle',  comment_char, ...
                       'CollectOutput', 1);
  f_data = raw_data{1};

  if isempty(f_data)
    error([char(10) '### There are no data. ' ...
                    'Did you use the right comment character?'            char(10) ...
                    '### The current comment character is: ' comment_char char(10) ...
                    '### Use a parameter list with the parameter:'        char(10) ...
                    '### plist(''comment_char'', ''%'')']);
  end

  fclose(fid);

  %%%%%%%%%%   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
      ao_data.name = sprintf('f_name_%02d', lauf);
      ao_data.xunits = xunits;
      ao_data.yunits = yunits;

      aa = init_ao;

      aa.name = sprintf('f_name_%02d', lauf);
      aa.data = ao_data;
      aa.hist = history(ALGONAME, VERSION, 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
      ao_data.name = sprintf('f_name_%02d', lauf);
      ao_data.xunits = xunits;
      ao_data.yunits = yunits;

      aa      = init_ao;
      aa.name = sprintf('f_name_%02d', lauf);
      aa.data = ao_data;
      aa.hist = history(ALGONAME, VERSION, 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 data object from the parameter list   %%%%%%%%%%
  for ii = 1:length(a)
    fields = fieldnames(a(ii).data);
    for jj = 1:length(pl.params)
      parameter = lower(pl.params(jj));
      if ismember(parameter.key, upper(fields))
        if iscell(parameter.val)
          if length(parameter.val) > 1
            if length(parameter.val) ~= length(a)
              warning([sprintf('The values of the key ''%s'' ', parameter.key) ...
                               'have not the same length as the lentgh of ' ...
                               'the output. Using the first value!']);
              a(ii).data.(lower(parameter.key)) = parameter.val{1};
            else
              a(ii).data.(lower(parameter.key)) = parameter.val{ii};
            end
          else
            a(ii).data.(lower(parameter.key)) = parameter.val{1};
          end
        else
          a(ii).data.(lower(parameter.key)) = parameter.val;
        end

      end % is parameter key member of the data object
    end % for-loop over all parameter
  end % for-loop over all ao's
```

```
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    aoFromSpecWin
%
% DESCRIPTION: Construct an ao from a Spectral window
%
% CALL:        a = aoFromSpecWin(a, win, VERSION, ALGONAME)
%
% PARAMETER:   a:         Initialised analysis object
%              win:       Spectral window object
%              VERSION:   cvs version string
%              ALGONAME: The m-file name (use the mfilename command)
%
% HISTORY:     07-05-2007 Hewitson
%              Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function a = aoFromSpecWin(a, win, VERSION, ALGONAME)

  % Make a cdata object
  ts = cdata(win.win);
  ts = set(ts, 'name', win.name);

  % Make an analysis object
  a.name = win.name;
  a.data = ts;
  a.hist = history(ALGONAME, VERSION, plist(param('win', win)));
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Default Parameter Lists
%
% CALL:        out = getDefaultPlist(set-string)
%
% PARAMETER:   set-string: A string which defines the default parameter list.
%
% HISTORY:     07-05-2007 Hewitson
%              Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function out = getDefaultPlist(varargin)

  % list of available parameter sets
  sets = {'From XML file', 'From ASCII file',...
    'Function', 'Values',...
```

```
    'Time-series Function', 'Frequency-series Function', ...
    'From Window', 'Waveform', 'From Repository',...
    'From Plist'};

  if nargin == 0
    out = sets;
    return
  end

  set = varargin{1};

  switch set
    case 'From Repository'
      out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
    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,...
        't0', time(0),...
        '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, 't0', time(0));
    case 'Frequency-series Function'
      out = plist('fsfcn', '1./f', 'f1', 1e-9, 'f2', 100, 'nf', 1000, 'scale', 'log', 'f', []);
    case 'From Window'
      out = plist('win', specwin('Hanning', 100));
    case 'From Plist'
      out = plist('Plist', plist);
    case 'Waveform'
      out = plist('waveform', 'sine wave',...
        'f', 1.23,...
        'phi', 0,...
        'fs', 10,...
        'nsecs', 10,...
        't0', time(0));
    otherwise
      out = plist();
  end

end
% END

end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

plist

PURPOSE ^

PLIST Plist class object constructor.

SYNOPSIS ^

function pl = plist(varargin)

DESCRIPTION ^

```
 PLIST Plist class object constructor.

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

 DESCRIPTION: PLIST Plist class object constructor.

 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('key1, val1, 'key2', 'val2') - create a plist with the
                                                 key/values pairs
       pl = plist(pl)                          - create a plist from a
                                                 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.


 VERSION:     $Id: plist.m,v 1.23 2008/02/24 10:15:56 hewitson Exp $

 HISTORY:     30-01-07 M Hewitson
                Creation
```

```
%%%
```

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: PLIST Plist class object constructor.
%
% 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('key1, val1, 'key2', 'val2') - create a plist with the
%                                                 key/values pairs
%       pl = plist(pl)                          - create a plist from a
%                                                 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.
%
%
% VERSION:     $Id: plist.m,v 1.23 2008/02/24 10:15:56 hewitson Exp $
%
% HISTORY:     30-01-07 M Hewitson
%                Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: plist.m,v 1.23 2008/02/24 10:15:56 hewitson Exp $';
CATEGORY = 'Constructor';

%%%%%   Check if this is a call for parameters   %%%%%
if nargin == 2
  if isa(varargin{1}, 'plist') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      pl = getDefaultPlist();
      return
    elseif strcmp(in, 'Version')
      pl = VERSION;
      return
    elseif strcmp(varargin{2}, 'Category')
      pl = CATEGORY;
      return
```

```
    end
  end
end

%%%%%%%%%%%%%%%%%%%%   define parameter properties list   %%%%%%%%%%%%%%%%%%%%

  function pl = init()
    pl.name     = 'plist object';
    pl.params   = [];
    pl.version  = VERSION;
    pl.created  = time;
    pl.plist    = '';
    pl = class(pl, 'plist');
  end

%%%%%%%%%%%%%%%%%%%%%%%   Create parameter list object   %%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%   pl = plist()   %%%%%%%%%%
% create default parameter list object
if nargin == 0

  pl = init();

elseif nargin == 1

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

    % Retrieve from repository?
    hostname = find(pli, 'hostname');
    if ~isempty(hostname)
      pl = retrievePlistFromRepository(pli);
    else
      pl = varargin{1};
    end

    % Store the input plist
    pl.plist = remove(pli, 'conn');

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

    pi = plstruct.params;
    params = [];
    for j=1:length(pi)
      if isstruct(pi(j))
        params = [params param(pi(j))];
      elseif isa(pi(j), 'param')
        params = [params pi(j)];
      else
```

```
        error ('### the parameter is not from class param.')
      end
    end

    pl         = init();
    pl.name    = plstruct.name;
    pl.created    = plstruct.created;
    if isstruct(pl.created)
      pl.created = time(pl.created);
    end
    pl.params  = params;
    pl.version = plstruct.version;

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

    filename = varargin{1};
    [path, name, ext, vers] = fileparts(filename);
    switch ext
      case '.mat'
        pl = load(filename);
        pl = pl.a;
      case '.xml'
        root_node = xmlread(filename);
        pl = ltpda_xmlread(root_node, 'plist');
      otherwise
        error('### Unknown file type.');
    end
    %%%%%%%%%%   pl = plist(param)   %%%%%%%%%%
  elseif isa(varargin{1}, 'param')
    pl        = init();

    %% Use only upper case characters for the 'key'
    param_list = varargin{1};
    for kk = 1:length(param_list)
      param_list(kk).key = upper(param_list(kk).key);
    end

    pl.params = param_list;

  else
    error ('### unknown arguments to construct a parameter list')
  end
elseif nargin == 2
  %%%%%%%%%%%   From DATABASE   %%%%%%%%%%%
  if isa(varargin{1}, 'database')

    pl = retrieve(varargin{1}, varargin{2:end});

    %%%%%%%%%%   pl = plist('key1', val1)   %%%%%%%%%%
  else
    pl  = init();
```

```
    key = upper(varargin{1});
    val =       varargin{2};

    pl.params  = param(key, val);
  end

elseif nargin > 2

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

    if ~ischar(key)
      error('### the key ''%s'' must be a character string', key)
    end

    found = 0;
    for ii = 1:length(params)
      if strcmpi(params(ii).key, key)
        % If the key exist in pl then do nothing, otherwise add the key value pair.
        warning('### Do not use the same key twice.');
        found = 1;
      end
    end

    if found == 0
      params = [params param(key, val)];
    end

  end

  pl         = init();
  pl.params  = params;

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

%-----------------------------------------------------------------------
% Retrieve a plist from the repository
%
  function       pls = retrievePlistFromRepository(pl, VERSION, ALGONAME)

    dpl = getDefaultPlist('From Repository');
    pl  = combine(pl, dpl);

    % 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);
    pls  = [];
    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 a plist
      if strcmp(tt, mfilename)
        %---- call database constructor
        a = ltpda_obj_retrieve(conn, id);
        %---- Add to output array
        pls = [pls a];
      else
        warning('    !skipping ID %d, type %s', id, tt);
      end

    end

    % close connection
    if closeConn
      close(conn);
    end
  end

%-----------------------------------------------------------------------
% Default Parameter Lists
%
  function out = getDefaultPlist(varargin)

    out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);

  end
```

```
end
% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

param

PURPOSE ^

PARAM Parameter object class constructor.

SYNOPSIS ^

function p = param(varargin)

DESCRIPTION ^

```
 PARAM Parameter object class constructor.

%%%
```

- save SAVE a param object to file.
- subsasgn SUBSASGN define index assignment for parameter properties.

SUBFUNCTIONS ^

- function p = init()

SOURCE CODE ^

```
function p = param(varargin)
% PARAM Parameter object class constructor.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: PARAM Parameter object class constructor.
%              Create a parameter object.
%
% CONSTRUCTOR: 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
%
% VERSION:     $Id: param.m,v 1.19 2008/02/13 17:33:38 mauro Exp $
%
% HISTORY:     30-01-2007 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: param.m,v 1.19 2008/02/13 17:33:38 mauro Exp $';
CATEGORY = 'Constructor';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'param') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      p = plist();
      return
    elseif strcmp(in, 'Version')
      p = VERSION;
      return
```

```
    elseif strcmp(in, 'Category')
      p = CATEGORY;
      return
    end
  end
end


%%%%%%%%%%%%%%%%%%%%%%%%   define parameter properties   %%%%%%%%%%%%%%%%%%%%%%%%

  function p = init()
    p.name    = 'param object';
    p.key     = '';
    p.val     = [];
    p.created = time;
    p.version = VERSION;
    p = class(p, 'param');
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%   Create parameter object   %%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%  p = param()   %%%%%%%%%%
% create default parameter object
if nargin == 0

  p = init();

elseif nargin == 1

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

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

    pl = varargin{1};
    key = upper(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

    p      = init();
    p.key  = key;
```

```
    p.val  = val;
    p.name = p.key;

    %%%%%%%%%%  p = param(struct)   %%%%%%%%%%
  elseif isstruct(varargin{1})

    p         = init();
    p.name    =       varargin{1}.name;
    p.key     = upper(varargin{1}.key);
    p.val     =       varargin{1}.val;
    if isstruct(varargin{1}.created)
      p.created = time(varargin{1}.created);
    end
    if isa(varargin{1}.created, 'time')
      p.created = varargin{1}.created;
    end
    p.version = varargin{1}.version;

  elseif ischar(varargin{1})

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

  else
    error('### unknown constructor type for param object.');
  end
elseif nargin == 2

  if isa(varargin{1}, 'database')

    p = retrieve(varargin{1}, varargin{2:end});

  %%%%%%%%%%  p = param(key, value)   %%%%%%%%%%
  else
    p         = init();
    p.key     = upper(varargin{1});
    p.val     =       varargin{2};
    p.name    = p.key;
  end
else
  error('### Unknown number of constructor arguments.');
end

```

```
0143 end % function p = param(varargin)
0144
0145
0146
0147
0148
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

specwin

PURPOSE ^

SPECWIN spectral window object class constructor.

SYNOPSIS ^

function w = specwin(varargin)

DESCRIPTION ^

```
 SPECWIN spectral window object class constructor.

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

 DESCRIPTION: SPECWIN spectral window object class constructor.
              Create a spectral window from libSpecWin.

  PROPERTIES:
       name     - 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
       nfft     - window length
       win      - window samples (column vector)

  CONSTRUCTORS:

       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.
       w = specwin('Types')           - return a list of all possible
                                        standard windows

 '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

 VERSION:     $Id: specwin.m,v 1.26 2008/02/24 10:15:55 hewitson Exp $

 HISTORY:     30-01-07 M Hewitson
                 Creation

%%%
```

```
%       flatness - window flatness
%       ws       - sum of window values
%       ws2      - sum of squares of window values
%       nfft     - window length
%       win      - window samples (column vector)
%
%  CONSTRUCTORS:
%
%       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.
%       w = specwin('Types')           - return a list of all possible
%                                        standard windows
%
% '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
%
% VERSION:     $Id: specwin.m,v 1.26 2008/02/24 10:15:55 hewitson Exp $
%
% HISTORY:     30-01-07 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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

ALGONAME = mfilename;
VERSION  = '$Id: specwin.m,v 1.26 2008/02/24 10:15:55 hewitson Exp $';
CATEGORY = 'Constructor';

% Check if this is a call for parameters
if (nargin == 2 || nargin == 3)
  if isa(varargin{1}, 'specwin') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      if nargin == 2
        w = getDefaultPlist();
      else
        w = getDefaultPlist(varargin{3});
      end
      return
    elseif strcmp(in, 'Version')
```

```
      w = VERSION;
      return
    elseif strcmp(in, 'Category')
      w = CATEGORY;
      return
    end
  end
end

%%%%%%%%%%%%%%%%%%%%   define parameter properties list   %%%%%%%%%%%%%%%%%%%%%

  function w = init()
    w.name     = 'no window';
    w.alpha    = -1;
    w.psll     = -1;
    w.rov      = -1;
    w.nenbw    = -1;
    w.w3db     = -1;
    w.flatness = -1;
    w.ws       = -1;
    w.ws2      = -1;
    w.win      = -1;
    w.plist    = '';
    w.created  = '1970-01-01 00:00:00';
    w.version  = VERSION;
    w = class(w, 'specwin');
  end

%%%%%%%%%%   spw = specwin()   %%%%%%%%%%
if nargin == 0

  w = init();

elseif nargin == 1
  %%%%%%%%%%   spw = specwin(specwin)   %%%%%%%%%%
  if isa(varargin{1}, 'specwin')
    w = varargin{1};

  %%%%%%%%%%   spw = specwin(struct)   %%%%%%%%%%
  elseif isstruct(varargin{1})

    w = class(varargin{1}, 'specwin');
    return

  %%%%%%%%%%   spw = specwin(plist)   %%%%%%%%%%
  elseif isa(varargin{1}, 'plist')

    pl    = varargin{1};
    wname = find(pl, 'Name');
    N     = find(pl, 'N');
    psll  = find(pl, 'PSLL');
    hostname = find(pl, 'hostname');
```

```

    if ~isempty(hostname)
      w = specwinFromRepository(pl, VERSION, ALGONAME);
    else
      switch wname
        case 'Kaiser'
          w = specwin(wname, N, psll);
        otherwise
          w = specwin(wname, N);
      end
    end

    w.plist = remove(varargin{1}, 'conn');

  elseif ischar(varargin{1})

    %%%%%%%%%%   spw = specwin('Types')   %%%%%%%%%%
    if strcmp(varargin{1}, 'Types')

      % return list of windows
      w = {...
        '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'...
        };

    %%%%%%%%%%   spw = specwin('foo.xml')   %%%%%%%%%%
    else

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

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

elseif nargin == 2
```

```

  %%%%%%%%%%   copy specwin   %%%%%%%%%%
  %%%%%%%%%%   spw = specwin(specwin, ...)   %%%%%%%%%%
  if isa(varargin{1}, 'specwin')
    w = varargin{1};

  %%%%%%%%%%   spw = specwin(database, ...)   %%%%%%%%%%
  elseif isa(varargin{1}, 'database')

    w = retrieve(varargin{1}, varargin{2:end});

  %%%%%%%%%%   create a particular type   %%%%%%%%%%
  %%%%%%%%%%   spw = specwin('Win_type', N)   %%%%%%%%%%
  elseif ischar(varargin{1})

    w = init();
    w = get_window(w, varargin{1}, varargin{2});

  else
    error('### unknown constructor for specwin object.')
  end

%%%%%%%%%%   create a Kaiser window   %%%%%%%%%%
%%%%%%%%%%   spw = specwin('Kaiser', N, psll)   %%%%%%%%%%
elseif nargin == 3

  w = init();
  w = get_window(w, varargin{1}, varargin{2}, varargin{3});

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



%-----------------------------------------------------------------------
% construct an ao from a repository
%
  function fs = specwinFromRepository(pl, VERSION, ALGONAME)

    dpl = getDefaultPlist('From Repository');
    pl  = combine(pl, dpl);

    % 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);
    fs  = [];
    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 a specwin
      if strcmp(tt, mfilename)
        %---- call database constructor
        a = ltpda_obj_retrieve(conn, id);
        %---- Add history
        %---- Add to output array
        fs = [fs a];
      else
        warning('    !skipping ID %d, type %s', id, tt);
      end

    end

    % close connection
    if closeConn
      close(conn);
    end

  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPL
%
% DESCRIPTION: Get default params
%
% HISTORY:     11-02-2008 M Hueller
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function out = getDefaultPlist(varargin)

    % list of available parameter sets
    sets = {'From Window Definition', 'From Repository'};
```

```

    if nargin == 0
      out = sets;
      return
    end

    set = varargin{1};
    switch set
      case 'From Repository'
        out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
      case 'From Window Definition'
        out = plist('Name', 'Kaiser', 'N', 10, 'PSLL', 150);
      otherwise
        out = plist();
    end

  end

end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

pole

PURPOSE ^

POLE construct a pole object.

SYNOPSIS ^

function p = pole(varargin)

DESCRIPTION ^

```
 POLE construct a pole object.

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

 DESCRIPTION:  POLE construct a pole object.

 CONSTRUCTORS: p = pole(f);
               p = pole(f,q);

 PARAMETERS:   'f'  - frequency
               'q'  - Q
               'ri' - complex value (column vector)

 VERSION:      $Id: pole.m,v 1.20 2008/02/17 11:38:51 hewitson Exp $

 HISTORY:      02-04-2007 M Hewitson
                  Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pole object into a string.

This function is called by:

- subsasgn SUBSASGN define index assignment for pole properties.

SUBFUNCTIONS ^

- function p = init()
- function p = poleFromPlist(p, pl)

SOURCE CODE ^

```
function p = pole(varargin)
% POLE construct a pole object.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION:  POLE construct a pole object.
%
% CONSTRUCTORS: p = pole(f);
%               p = pole(f,q);
%
% PARAMETERS:   'f'  - frequency
%               'q'  - Q
%               'ri' - complex value (column vector)
%
% VERSION:      $Id: pole.m,v 1.20 2008/02/17 11:38:51 hewitson Exp $
%
% HISTORY:      02-04-2007 M Hewitson
%                  Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: pole.m,v 1.20 2008/02/17 11:38:51 hewitson Exp $';
CATEGORY = 'Constructor';

%%%%%   Check if this is a call for parameters   %%%%%
if nargin == 2
  if isa(varargin{1}, 'pole') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      p = plist;
      return
    elseif strcmp(in, 'Version')
      p = VERSION;
      return
    elseif strcmp(in, 'Category')
      p = CATEGORY;
      return
    end
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%   define pole properties   %%%%%%%%%%%%%%%%%%%%%%%%%

  function p = init()
    p.name    = 'not defined';
    p.f       = NaN;
```

```
    p.q       = NaN;
    p.ri      = NaN;
    p.version = VERSION;
    p.created = time;
    p.plist   = plist;
    p         = class(p, 'pole');
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%   Create pole object   %%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%   p = pole()   %%%%%%%%%%
if nargin == 0

  p = init();

elseif nargin == 1

  %%%%%%%%%% From File %%%%%%%%%%%%
  if ischar(varargin{1})

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

  %%%%%%%%%%   p = pole(number)            %%%%%%%%%%
  %%%%%%%%%%   p = pole(complex number)   %%%%%%%%%%
  elseif isnumeric(varargin{1})

    p = init();

    if isreal(varargin{1})
      p.name    = 'real pole';
      p.f       = varargin{1};
      p.q       = NaN;
      p.ri      = pfq2ri(p.f);
    else
      p.name     = 'complex pole';
      [p.f, p.q] = pri2fq(varargin{1});
      if length(varargin{1}) == 1
        p.ri      = [varargin{1}; conj(varargin{1})];
      elseif length(varargin{1}) == 2
        p.ri      = varargin{1};
      else
```

```
        error('### Unknown constructor method. Please specify a complex number or a complex conjugate pair.');
      end
    end

  %%%%%%%%%%   p = pole(struct)   %%%%%%%%%%
  elseif isstruct(varargin{1})

    p = init();

    fields = fieldnames(varargin{1});
    for ii = 1:length(fields)
      field = fields{ii};
      try
        p.(field) = varargin{1}.(field);
      catch
          error('### The field ''%s'' in the struct is not a pole property.', field)
      end
    end

  %%%%%%%%%%   p = pole(plist)   %%%%%%%%%%
  elseif isa(varargin{1}, 'plist')

    p = init();
    p = poleFromPlist(p, varargin{1});
    p.plist = varargin{1};

  %%%%%%%%%%   p = pole(pole)   %%%%%%%%%%
  elseif isa(varargin{1}, 'pole')
    p = varargin{1};
  else
    error('### unknown constructor method for pole class.');
  end

elseif nargin == 2
  %%%%%%%%%%% From DATABASE
  if isa(varargin{1}, 'database')
    p = retrieve(varargin{1}, varargin{2:end});
  %%%%%%%%%%   p = pole(f, q)   %%%%%%%%%%
  elseif isnumeric(varargin{1}) && isnumeric(varargin{2})

    ri   =  pfq2ri(varargin{1}, varargin{2});

    if length(ri) == 2 && isreal(ri(1)) && ri(1) == ri(2)
      disp('- splitting to two real poles');
      p      =  init();
      if isnan(varargin{2})
        p.name = 'real pole';
      else
        p.name = 'complex pole';
      end
```

```
      p.f    =  varargin{1};
      p.q    =  varargin{2};
      p.ri   = ri;
      p      = [p p];
    else
      p      =  init();
      if isnan(varargin{2})
        p.name = 'real pole';
      else
        p.name = 'complex pole';
      end
      p.f    =  varargin{1};
      p.q    =  varargin{2};
      p.ri   = ri;
    end
  else
    error('### unknown constructor method for pole class.');
  end

end

%-----------------------------------------------------------------------
% construct a pole from plist
%
function p = poleFromPlist(p, pl)

  f  = find(pl, 'f');
  q  = find(pl, 'q');
  ri = find(pl, 'ri');

  name    = find(pl, 'name');
  version = find(pl, 'version');
  created = find(pl, 'created');

  if ~isempty(f)
    if ~isempty(q)
      % complex pole
      if isnan(q)
        p.name    = 'real pole';
      else
        p.name    = 'complex pole';
      end
      p.f       = f;
      p.q       = q;
      p.ri      = pfq2ri(p.f, p.q);

    else
      % real pole
      p.name    = 'real pole';
      p.f       = f;
      p.q       = NaN;
      p.ri      = pfq2ri(p.f);
```

```

    end
  elseif ~isempty(ri)
    % complex pole
    p.name    = 'complex pole';
    [p.f, p.q] = pri2fq(ri);
    p.ri      = [ri; conj(ri)];

  else
    error('### unknown constructor method for pole class.');
  end

  % Set other the properties in the plist
  if ~isempty(name)
    p.name = name;
  end
  if ~isempty(version)
    p.version = version;
  end
  if ~isempty(created)
    p.created = created;
  end


end

end % function p = pole(varargin)

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

zero

PURPOSE ^

ZERO construct a pole object.

SYNOPSIS ^

function p = zero(varargin)

DESCRIPTION ^

```
 ZERO construct a pole object.

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

 DESCRIPTION:  ZERO construct a pole object.

 CONSTRUCTORS: z = zero(f);      - frequency (Hz)
               z = zero(f,q);    - frequency (Hz), Q
               z = zero(c);      - complex number
               z = zero(pl);     - plist()

 PARAMETERS:   'f'  - frequency
               'q'  - Q
               'ri' - complex value (column vector)

 VERSION:      $Id: zero.m,v 1.20 2008/02/17 11:38:51 hewitson Exp $

 HISTORY:      02-04-2007 Hewitson
                  Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a zero object into a string.

This function is called by:

- subsasgn SUBSASGN define index assignment for zero properties.

SUBFUNCTIONS ^

- function p = init()
- function p = zeroFromPlist(p, pl)

SOURCE CODE ^

```
function p = zero(varargin)
% ZERO construct a pole object.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION:  ZERO construct a pole object.
%
% CONSTRUCTORS: z = zero(f);       - frequency (Hz)
%               z = zero(f,q);     - frequency (Hz), Q
%               z = zero(c);       - complex number
%               z = zero(pl);      - plist()
%
% PARAMETERS:   'f'  - frequency
%               'q'  - Q
%               'ri' - complex value (column vector)
%
% VERSION:      $Id: zero.m,v 1.20 2008/02/17 11:38:51 hewitson Exp $
%
% HISTORY:      02-04-2007 Hewitson
%                  Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: zero.m,v 1.20 2008/02/17 11:38:51 hewitson Exp $';
CATEGORY = 'Constructor';

%%%%%   Check if this is a call for parameters   %%%%%
if nargin == 2
  if isa(varargin{1}, 'zero') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      p = plist;
      return
    elseif strcmp(in, 'Version')
      p = VERSION;
      return
    elseif strcmp(in, 'Category')
      p = CATEGORY;
      return
    end
  end
end

```

```
%%%%%%%%%%%%%%%%%%%%%%%%   define zero properties   %%%%%%%%%%%%%%%%%%%%%%%%

  function p = init()
    p.name    = 'not defined';
    p.f       = NaN;
    p.q       = NaN;
    p.ri      = NaN;
    p.version = VERSION;
    p.created = time;
    p.plist   = plist;
    p         = class(p, 'zero');
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%   Create zero object   %%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%   p = zero()   %%%%%%%%%%
if nargin == 0

  p = init();

elseif nargin == 1

  %%%%%%%%%%% From File %%%%%%%%%%%%%%%%
  if ischar(varargin{1})

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

  %%%%%%%%%%   p = zero(number)           %%%%%%%%%%
  %%%%%%%%%%   p = zero(complex number)   %%%%%%%%%%
  elseif isnumeric(varargin{1})

    p = init();

    if isreal(varargin{1})
      p.name    = 'real zero';
      p.f       = varargin{1};
      p.q       = NaN;
      p.ri      = pfq2ri(p.f);
    else
      p.name     = 'complex zero';
      [p.f, p.q] = pri2fq(varargin{1});
```

```
      if length(varargin{1}) == 1
        p.ri      = [varargin{1}; conj(varargin{1})];
      elseif length(varargin{1}) == 2
        p.ri      = varargin{1};
      else
        error('### Unknown constructor method. Please specify a complex number or a complex conjugate pair.');
      end
    end

  %%%%%%%%%%   p = zero(struct)   %%%%%%%%%%
  elseif isstruct(varargin{1})

    p = init();

    fields = fieldnames(varargin{1});
    for ii = 1:length(fields)
      field = fields{ii};
      try
        p.(field) = varargin{1}.(field);
      catch
          error('### The field ''%s'' in the struct is not a zero property.', field)
      end
    end

  %%%%%%%%%%   p = zero(plist)   %%%%%%%%%%
  elseif isa(varargin{1}, 'plist')

    p = init();
    p = zeroFromPlist(p, varargin{1});

    % Store input plist
    p.plist = varargin{1};

  %%%%%%%%%%   p = zero(zero)   %%%%%%%%%%
  elseif isa(varargin{1}, 'zero')
    p = varargin{1};

  else
    error('### unknown constructor method for zero class.');
  end

elseif nargin == 2
  %%%%%%%%%%% From DATABASE
  if isa(varargin{1}, 'database')
    p = retrieve(varargin{1}, varargin{2:end});
  %%%%%%%%%%   p = zero(f, q)   %%%%%%%%%%
  elseif isnumeric(varargin{1}) && isnumeric(varargin{2})

    ri   =  pfq2ri(varargin{1}, varargin{2});

```

```
    if length(ri) == 2 && isreal(ri(1)) && ri(1) == ri(2)
      disp('- splitting to two real zeros');
      p      =  init();
      if isnan(varargin{2})
        p.name = 'real zero';
      else
        p.name = 'complex zero';
      end
      p.f    =  varargin{1};
      p.q    =  varargin{2};
      p.ri   = ri;
      p      = [p p];
    else
      p      =  init();
      if isnan(varargin{2})
        p.name = 'real zero';
      else
        p.name = 'complex zero';
      end
      p.f    =  varargin{1};
      p.q    =  varargin{2};
      p.ri   = ri;
    end

  else
    error('### unknown constructor method for zero class.');
  end

end

%-------------------------------------------------------------------------
% construct a zero from plist
%
function p = zeroFromPlist(p, pl)

  f  = find(pl, 'f');
  q  = find(pl, 'q');
  ri = find(pl, 'ri');

  name    = find(pl, 'name');
  version = find(pl, 'version');
  created = find(pl, 'created');

  if ~isempty(f)
    if ~isempty(q)
      % complex zero
      if isnan(q)
        p.name    = 'real zero';
      else
        p.name    = 'complex zero';
      end
      p.f       = f;
```

```
      p.q       = q;
      p.ri      = pfq2ri(p.f, p.q);

    else
      % real zero
      p.name    = 'real zero';
      p.f       = f;
      p.q       = NaN;
      p.ri      = pfq2ri(p.f);

    end
  elseif ~isempty(ri)
    % complex zero
    p.name    = 'complex zero';
    [p.f, p.q] = pri2fq(ri);
    p.ri      = [ri; conj(ri)];

  else
    error('### unknown constructor method for zero class.');
  end

  % Set other the properties in the plist
  if ~isempty(name)
    p.name = name;
  end
  if ~isempty(version)
    p.version = version;
  end
  if ~isempty(created)
    p.created = created;
  end

end


end % function p = zero(varargin)

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

pzmodel

PURPOSE ^

PZMODEL constructor for pzmodel class.

SYNOPSIS ^

function pzm = pzmodel(varargin)

DESCRIPTION ^

```
 PZMODEL constructor for pzmodel class.

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

 DESCRIPTION: PZMODEL constructor for pzmodel class.

 CONSTRUCTOR: pzm = pzmodel(g, p, z)          % construct from gain, poles, zeros
              pzm = pzmodel(g, p, z, 'name')  % construct including name
              pzm = pzmodel(pl);              % construct from plist
              pzm = pzmodel('foo.fil');       % construct from LISO .fil file


 PARAMETERS:  'gain'     - model gain
              'poles'    - vector of pole objects
              'zeros'    - vector of zero objects
              'name'     - name of model
              'filename' - name of LISO .fil file

 VERSION:     $Id: pzmodel.m,v 1.21 2008/02/24 10:15:55 hewitson Exp $

 HISTORY:     03-04-2007 M Hewitson
                 Creation

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = pzmodel(pzmodel,'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = pzmodel(pzmodel,'Version')

 The following call returns a string that contains the routine category:

 >> category = pzmodel(pzmodel,'Category')
```

```
%%%
```

```
%                 Creation
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = pzmodel(pzmodel,'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = pzmodel(pzmodel,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = pzmodel(pzmodel,'Category')
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ALGONAME = mfilename;
VERSION  = '$Id: pzmodel.m,v 1.21 2008/02/24 10:15:55 hewitson Exp $';
CATEGORY = 'Constructor';

%%%%%   Check if this is a call for parameters   %%%%%
if (nargin == 2 || nargin == 3)
  if isa(varargin{1}, 'pzmodel') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmpi(in, 'Params')
      if nargin == 2
        pzm = getDefaultPlist();
      else
        pzm = getDefaultPlist(varargin{3});
      end
      return
    elseif strcmpi(in, 'Version')
      pzm = VERSION;
      return
    elseif strcmpi(in, 'Category')
      pzm = CATEGORY;
      return
    end
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%   define pzmodel properties   %%%%%%%%%%%%%%%%%%%%%%%%%

  function pz = init()
    pz.name    = 'pzmodel object';
    pz.gain    = 0;
    pz.poles   = pole;
    pz.zeros   = zero;
    pz.created = time;
    pz.version = VERSION;
    pz.plist   = '';
```

```
    pz         = class(pz, 'pzmodel');
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%   Create pzmodel object   %%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%   pzm = pzmodel()   %%%%%%%%%%
if nargin == 0

  pzm = init();

elseif nargin == 1

  %%%%%%%%%%   p = pzmodel(pzmodel)   %%%%%%%%%%
  if isa(varargin{1}, 'pzmodel')
    pzm = varargin{1};

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

    filename = varargin{1};
    [path, name, ext, vers] = fileparts(filename);
    switch ext
      case '.fil'
        pzm = pzmFromLISO(filename);
      case '.mat'
        pzm = load(filename);
        pzm = pzm.a;
      case '.xml'
        root_node = xmlread(filename);
        pzm = ltpda_xmlread(root_node, 'pzmodel');
      otherwise
        error('### Unknown file type.');
    end

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

    pzm = init();

    fields = fieldnames(varargin{1});
    for ii = 1:length(fields)
      field = fields{ii};

      %%% pole -> pole-object
      if strcmp(field, 'poles')

        ps = varargin{1}.poles;
        poles = [];
        for jj = 1:length(ps)
          if isstruct(ps(jj))
            poles = [poles pole(ps(jj))];
          else
```

```
            poles = [poles ps(jj)];
          end
        end
        pzm.poles = poles;

        %%% zero -> pole-object
      elseif strcmp(field, 'zeros')

        zs = varargin{1}.zeros;
        zeros = [];
        for jj = 1:length(zs)
          if isstruct(zs(jj))
            zeros = [zeros zero(zs(jj))];
          else
            zeros = [zeros zs(jj)];
          end
        end
        pzm.zeros = zeros;

        %%% created -> time-object
      elseif strcmp(field, 'created')
        if isstruct(varargin{1}.created)
          pzm.created = time(varargin{1}.created);
        else
          pzm.created = varargin{1}.created;
        end
        %%% All other
      else
        try
          pzm.(field) = varargin{1}.(field);
        catch
          error('### The field ''%s'' in the struct is not a pole property.', field)
        end
      end
    end

    %%%%%%%%%%   pzm = pzmodel(plist)   %%%%%%%%%%
  else
    pl  = varargin{1};

    filename    = find(pl, 'filename');
    hostname    = find(pl, 'hostname');

    if ~isempty(filename)

      pzm = pzmodel(filename);

    elseif ~isempty(hostname)

      pzm = pzmodelFromRepository(pl, VERSION, ALGONAME);

    else
```

```
0179
0180        pzm = init();
0181
0182        dpl = getDefaultPlist('From Poles/Zeros');
0183        pl  = combine(pl, dpl);
0184        pzm.gain = find(pl, 'gain');
0185        ps  = find(pl, 'poles');
0186        pso=[];
0187        for j=1:length(ps)
0188            if ~isnan(ps(j).f)
0189                pso = [pso ps(j)];
0190            end
0191        end
0192        pzm.poles = pso;
0193        zs = find(pl, 'zeros');
0194        zso=[];
0195        for j=1:length(zs)
0196            if ~isnan(zs(j).f)
0197                zso = [zso zs(j)];
0198            end
0199        end
0200        pzm.zeros = zso;
0201
0202        % If the following properties are not set in the plist then use the
default
0203        % values
0204        if ~isempty(find(pl, 'version'))
0205          pzm.version = find(pl, 'version');
0206        end
0207        if ~isempty(find(pl, 'created'))
0208          pzm.created = find(pl, 'created');
0209        end
0210        if ~isempty(find(pl, 'name'))
0211          pzm.name = find(pl, 'name');
0212        end
0213      end
0214
0215      % Store the input plist
0216      pzm.plist = remove(pl, 'conn');
0217
0218    end
0219
0220 elseif nargin == 2
0221    %%%%%%%%%%%   From DATABASE   %%%%%%%%%%%
0222    if isa(varargin{1}, 'database')
0223      pzm = retrieve(varargin{1}, varargin{2:end});
0224    else
0225      error('### incorrect pzmodel constructor.');
0226    end
0227
0228
0229    %%%%%%%%%%%   3 or 4 inputs                                %%%%%%%%%%%
0230    %%%%%%%%%%%   pzm = pzmodel(gain, poles, zeros)            %%%%%%%%%%%
```

```
  %%%%%%%%%%%   pzm = pzmodel(gain, poles, zeros, name)   %%%%%%%%%%%
  %%%%%%%%%%%   pzm = pzmodel(ao, poles, zeros)           %%%%%%%%%%%
  %%%%%%%%%%%   pzm = pzmodel(ao, poles, zeros, name)     %%%%%%%%%%%
elseif nargin >= 3

  pzm = init();

  a  = varargin{1};
  if isa(a, 'ao')
    gain = a.data.y(1);
  else
    gain = a;
  end
  poles = varargin{2};
  zeros = varargin{3};
  if nargin == 4
    name = char(varargin{4});
  else
    name = 'pzmodel';
  end

  % construct pz model object
  pzm.name  = name;
  pzm.gain  = gain;
  pzm.poles = poles;
  pzm.zeros = zeros;
  pzm.plist = plist('gain', gain, 'poles', poles, 'zeros', zeros);

else
  error('### incorrect number of inputs.');
end

%-----------------------------------------------------------------------
% construct a pzmodel from a repository
%
  function aos = pzmodelFromRepository(pl, VERSION, ALGONAME)

    dpl = getDefaultPlist('From Repository');
    pl  = combine(pl, dpl);

    % 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);
    aos  = [];
    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
        a = ltpda_obj_retrieve(conn, id);
        %---- Add history
        a.plist = pl;
        %---- Add to output array
        aos = [aos a];
      else
        warning('    !skipping ID %d, type %s', id, tt);
      end

    end

    % close connection
    if closeConn
      close(conn);
    end

  end

%-----------------------------------------------------------------------
%
% Function to read a pole/zero model from a LISO file
%
  function pzm = pzmFromLISO(filename)

    pzm = init();

    poles = [];
    zeros = [];
    gain  = [];

    % Open the file for reading
    fd = fopen(filename, 'r');
    disp(['--- 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 = pole(pf);
          disp(sprintf('   -- found pole: %g', pf));
        else
          p = pole(pf, pq);
          disp(sprintf('   -- 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 = zero(zf);
          disp(sprintf('   -- found zero: %g', zf));
        else
          z = zero(zf, zq);
          disp(sprintf('   -- 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);
        disp(sprintf('   -- found factor: %g', gain));
      end

    end

    % Close file
    fclose(fd);

    % get model name
    [path, name, ext, vers] = fileparts(filename);

    pzm = pzmodel(gain, poles, zeros, name);
    pzm.plist = plist('filename', filename);

  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:    getDefaultPL
%
% DESCRIPTION: Get default params
%
% HISTORY:     11-02-2008 M Hueller
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function out = getDefaultPlist(varargin)

    % list of available parameter sets
    sets = {'From Poles/Zeros', 'From LISO file', 'From Repository'};

    if nargin == 0
      out = sets;
```

```
      return
    end

    set = varargin{1};
    switch set
      case 'From Repository'
        out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
      case 'From LISO file'
        out = plist('filename', 'foo.fil');
      case 'From Poles/Zeros'
        out = plist('gain', 1, ...
          'poles', pole, ...
          'zeros', zero, ...
          'name', '');
      otherwise
        out = plist();
    end

  end
end
% END


```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

< Master index Index for classes/@ao >

Index for classes/@ao

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: ao %%%%%%%%%%%%%%%%%%%%%%
abs	ABS overloads the Absolute value operator for Analysis objects.
acos	ACOS overloads the acos operator for Analysis objects. Inverse cosine result in radians.
ao	AO analysis object class constructor.
ao2m	AO2M converts an analysis object to an '.m' file based on the history contained in the analysis object.
asin	ASIN overloads the asin operator for Analysis objects. Inverse sine result in radians.
atan	ATAN overloads the atan operator for Analysis objects. Inverse tangent result in radians.
attachm	ATTACHM attach the m file being executes to the analysis object.
attachmdl	ATTACHMDL attach an mdl file to the analysis object.
cat	CAT concatonate AOs into a vector.
char	CHAR overloads char() function for analysis objects.
complex	COMPLEX overloads the complex operator 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.
ctranspose	CTRANSPOSE overloads the ' operator for Analysis Objects.
demux	DEMUX splits the input vector of AOs into a number of output AOs.
det	DET overloads the determinant function for Analysis objects.
diag	DIAG overloads the diagonal operator for Analysis Objects.
display	DISPLAY implement terminal display for analysis object.
downsample	DOWNSAMPLE AOs containing time-series data.
eig	EIG overloads the determinant function for Analysis objects.
eq	EQ overloads the == operator 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
extractmdl	EXTRACTMDL extracts an mdl file from an analysis object and saves it to
fft	FFT overloads the fft operator for Analysis objects.
filter	FILTER overides the filter function for analysis objects.
filtfilt	FILTFILT overides the filtfilt function for analysis objects.
find	FIND particular samples that satisfy the input query and return a new AO.
ge	GE overloads >= operator for analysis objects. Compare the y-axis values.
get	GET get ao properties.
gt	GT overloads > operator for analysis objects. Compare the y-axis values.
hist	HIST overloads the histogram function (hist) of MATLAB 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 specified by the input parameter list.
inv	INV overloads the inverse function for Analysis Objects.
iplot	IPLOT provides an intelligent plotting tool for LTPDA.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given ao has all the correct fields of the correct type.
join	JOIN multiple AOs into a single AO.
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.
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.
lt	LT overloads < operator for analysis objects. Compare the y-axis values.
md5	MD5 computes an MD5 checksum from an analysis objects.
mean	MEAN overloads the mean operator for Analysis objects. Compute the mean value.

median	MEDIAN overloads the median operator for Analysis objects. Compute the median value.
minus	MINUS overloads - operator for analysis objects.
mpower	MPOWER implements power operator for analysis objects.
mrdivide	MRDIVIDE overloads / operator for analysis objects.
mtimes	MTIMES overloads * operator for analysis objects.
mux	MUX concatonate AOs into a vector.
ne	NE overloads the ~= operator for analysis objects.
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 two analysis objects.
polyfit	POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
power	POWER implements power operator for analysis objects.
rdivide	RDIVIDE overloads ./ operator for analysis objects.
real	REAL overloads the real operator for Analysis objects.
resample	RESAMPLE overloads resample function for AOs.
save	SAVE overloads save operator for analysis objects.
select	SELECT select particular samples from the input AOs and return new AOs with only those samples.
set	SET set an analysis object property.
setnh	SET set an analysis object property without recording the history.
sin	SIN overloads the sin operator for Analysis objects. Sine of argument in radians.
spectrogram	SPECTROGRAM computes a spectrogram of the given ao/tsdata.
split	SPLIT split an analysis object into the specified segments.
sqrt	SQRT overloads the sqrt operator for Analysis objects. Square root.
stairs	STAIRS overloads the stairs function for Analysis Objects.
std	STD overloads the std operator for Analysis objects. Compute the standard deviation.
string	STRING writes a command string that can be used to recreate the input Analysis object(s).
subsasgn	SUBSASGN define index assignment for analysis object properties.

subsref	SUBSREF Define field name indexing for analysis objects.
sum	SUM overloads the sum operator for Analysis objects. Compute the sum value.
svd	SVD overloads the determinant function for Analysis objects.
tan	TAN overloads the tan operator for Analysis objects. Tangent of argument in radians.
times	TIMES overloads .* 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.
upsample	UPSAMPLE overloads upsample function for AOs.
validate	VALIDATE checks that the input Analysis Object is reproducible and valid.
var	VAR overloads the var operator for Analysis objects. Compute the variance.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

Matlab Index

Matlab Directories

- classes/@ao
- classes/@ao/private
- classes/@cdata
- classes/@cdata/private
- classes/@fsdata
- classes/@fsdata/private
- classes/@history
- classes/@history/private
- classes/@mfir
- classes/@mfir/private
- classes/@miir
- classes/@miir/private
- classes/@param
- classes/@param/private
- classes/@plist
- classes/@plist/private
- classes/@pole
- classes/@pole/private
- classes/@provenance
- classes/@provenance/private
- classes/@pzmodel
- classes/@pzmodel/private
- classes/@specwin
- classes/@specwin/private
- classes/@time
- classes/@time/private
- classes/@timeformat
- classes/@timeformat/private
- classes/@timespan
- classes/@timespan/private
- classes/@tsdata
- classes/@tsdata/private
- classes/@xydata
- classes/@xydata/private
- classes/@xyzdata
- classes/@zero
- classes/@zero/private
- m/etc
- m/gui/ao_browser
- m/gui/ltpdaRepoGUI
- m/gui/ltpdaRepoGUI/callbacks
- m/gui/pzmodel_designer

- m/helper
- m/helper/generic_class_fcn
- m/math
- m/mysql
- m/noisegenerator
- m/noisegenerator/private
- m/plottools
- m/sigproc/frequency_domain
- m/sigproc/time_domain
- m/timetools
- m/timetools/statespacefunctions
- mex

Matlab Files found in these Directories

Contents	find	ltpda_saveobj	save
Contents	format	ltpda_simulate_discrete	save
Contents	fq2fac	ltpda_spikecleaning	save
Contents	fsdata	ltpda_ss_assemble	save
Contents	gcbsh	ltpda_ss_assemble_arrange	save
Contents	ge	ltpda_ss_assemble_fusion	save
Contents	generic_eq	ltpda_ss_check	save
Contents	generic_get	ltpda_ss_iir2ss	save
Contents	generic_set	ltpda_ss_kalman_filter	save
Contents	get	ltpda_ss_make	select
Contents	get	ltpda_ss_modify	set
Contents	get	ltpda_ss_modify_time_setup	set
Contents	get	ltpda_ss_pz2ss	set
Contents	get	ltpda_ss_ss2iir	set
Contents	get	ltpda_startup	set
Contents	get	ltpda_str2cells	set
Contents	get	ltpda_strs2cells	set
Contents	get	ltpda_structcat	set
Contents	get	ltpda_structidx	set
Contents	get	ltpda_tfe	set
Contents	get	ltpda_timedomainfit	set
Contents	get	ltpda_uitree	set
Contents	get	ltpda_uitreenode	set
Contents	get	ltpda_unwrapdeg	set
Contents	get	ltpda_user_classes	set
Contents	get	ltpda_user_fcn_template	set

Contents	get	ltpda_utc2gps	set
Contents	get	ltpda_versions	set
Contents	getFields	ltpda_xaxis	set
Contents	getNodes	ltpda_xcorr	setFreq
Contents	getNodes_plot	ltpda_xml_read_datasamples	setTime
Contents	getTables	ltpda_xmlread	setWorkspaceObjsList
Contents	getWorkspaceObjs	ltpda_xmlwrite	setWorkspaceObjsListQL
Contents	get_curr_m_file_path	ltpda_xscale	set_xy_axis
Contents	get_window	ltpda_xspec	set_xy_axis
Contents	get_xy_values	ltpda_xticks	set_xy_axis
Contents	get_xy_values	ltpda_yaxis	set_xy_axis
Contents	get_xy_values	ltpda_yscale	setnh
Contents	get_xy_values	ltpda_yticks	sin
Contents	get_xyz_values	ltpda_zaxis	single_operation
Contents	getlowerFreq	ltpda_zscale	single_operation
Contents	getupperFreq	md5	single_operation
Contents	gt	mean	single_operation
Contents	hist	median	single_operation
Contents	hist2dot	method_fcn_template	spectrogram
Contents	hist2m	mfiltload	specwin
Contents	history	mfir	split
Contents	imag	miir	sqlResultsGUI
Contents	index	minus	sqrt
Contents	interp	minus	stairs
abcascade	inv	mkbandpass	std
abs	iplot	mkbandpass	string
acos	isfield	mkbandreject	string
ao	isfield	mkbandreject	string
ao2m	isfield	mkhighpass	string
ao_fcn_template	isfield	mkhighpass	string
aooperate	isfield	mklowpass	string
append	isfield	mklowpass	string
asin	isfield	mpower	string
atan	isfield	mrdivide	string
attachm	isfield	mtimes	string
attachmdl	isfield	muitable	string
buildConditions	isfield	mux	string
buildquery	isfield	mux	string
cat	isfield	mysql_connect	string

cat	isfield	mysql_getMD5hash	strpad
cat	isfield	mysql_getMaxId	subsasgn
cat	isfield	mysql_getObjIds	subsasgn
cb_classMethodList	isfield	mysql_getObjType	subsasgn
cb_condPlusBtn	isfield	mysql_getUserID	subsasgn
cb_condSubtractBtn	isfield	mysql_getXdoc	subsasgn
cb_connectBtn	islinespec	ne	subsasgn
cb_dbSelect	isparam	ne	subsasgn
cb_executeQuery	isvalid	ne	subsasgn
cb_fieldSelect	isvalid	ne	subsasgn
cb_getDBs	isvalid	ne	subsasgn
cb_mainfigClose	isvalid	ne	subsasgn
cb_refreshObjsList	isvalid	ne	subsasgn
cb_refreshObjsListQL	isvalid	ne	subsasgn
cb_retrieveBtn	isvalid	ne	subsasgn
cb_serverSelect	isvalid	ne	subsasgn
cb_submitBtn	isvalid	ne	subsasgn
cb_tableSelect	isvalid	ne	subsasgn
cb_workspaceObjsList	isvalid	ne	subsasgn
cb_workspaceObjsListQL	isvalid	ne	subsasgn
cdata	isvalid	ne	subsref
char	isvalid	ne	subsref
char	isvalid	ne	subsref
char	isvalid	ne	subsref
char	isvalid	ne	subsref
char	isvalid	ngconv	subsref
char	isvalid	nginit	subsref
char	join	ngprop	subsref
char	kaiser_alpha	ngsetup	subsref
char	kaiser_flatness	norm	subsref
char	kaiser_nenbw	nparams	subsref
char	kaiser_rov	param	subsref
char	kaiser_w3db	parseFilterParams	subsref
char	le	parseFilterParams	subsref
char	len	pfq2ri	subsref
char	ln	pfq2ri	subsref
char	log	phase	subsref
char	log10	phase	subsref
char	logindlg	plist	subsref

collect_inputs
combine
complex
compute_interval
conj
conv_noisegen
cos
cp2iir
ctranspose
cz2iir
demux
det
diag
display
display
display
display
display
display
display
display
display
display
display
display
display
display
display
downsample
eig
eq
eq
eq
eq
lt
ltpda_allgrid
ltpda_allxaxis
ltpda_allxlabel
ltpda_allxscale
ltpda_allyaxis
ltpda_allylabel
ltpda_allyscale
ltpda_arma_freq
ltpda_arma_time
ltpda_chkts
ltpda_classes
ltpda_cohere
ltpda_compute_f
ltpda_cpsd
ltpda_cscale
ltpda_dbquery
ltpda_deg2rad
ltpda_dft
ltpda_disp
ltpda_fields2list
ltpda_filescan
ltpda_fq2ri
ltpda_fq2ri2
ltpda_funchash
ltpda_gapfilling
ltpda_getAOsInTimeSpan
ltpda_getmax
ltpda_gps2utc
ltpda_hash
ltpda_insert
ltpda_io_collect
ltpda_isobject
ltpda_isuserobject
ltpda_jacobian
ltpda_javacomponent
ltpda_kalman_filter
ltpda_label
ltpda_lcohere
plot
plot
plot
plus
plus
pole
polydetrend
polyfit
power
pri2fq
pri2fq
provenance
pset
pzm2ab
pzmodel
pzmodel_helper
pzmresp
rdivide
real
redesign
redesign
remove
repogui
resample
reshapeF
reshapeT
reshapeX
resp
resp
resp
resp
resp
retrieve
retrieve
retrieve
retrieve
retrieve
sum
svd
tan
time
timeformat
times
timeshift
timespan
tomfir
tomiir
transpose
tsdata
uitabpanel
upsample
utp_ltpda_simulate_discrete
utp_ltpda_ss_assemble_1
utp_ltpda_ss_assemble_2
utp_ltpda_ss_iir2ss
utp_ltpda_ss_kalman_filter
utp_ltpda_ss_make
utp_ltpda_ss_modify
utp_ltpda_ss_modify_time_setup
utp_ltpda_ss_pz2ss
utp_ltpda_ss_random_generate
utp_ltpda_ss_ss2iir
validate
var
win_BH92
win_Bartlett
win_FTHP
win_FTNI
win_FTSRS
win_HFT116D
win_HFT144D
win_HFT169D
win_HFT196D
win_HFT223D
win_HFT248D
win_HFT70

eq
eq
eq
eq
eq
eq
eq
eq
eq
eq
eq
exp
explore_ao
export
extractm
extractmdl
fft
fillClassMethodsList
filload
filter
filtfilt
find
ltpda_lcpsd
ltpda_legendAdd
ltpda_lincom
ltpda_linedetect
ltpda_lpsd
ltpda_ltf_plan
ltpda_ltfe
ltpda_lxspec
ltpda_mat2str
ltpda_matchvectors
ltpda_msuptitle
ltpda_nfest
ltpda_noisegen
ltpda_num2str
ltpda_obj_retrieve
ltpda_obj_submit
ltpda_phase
ltpda_polydetrend
ltpda_polyreg
ltpda_psd
ltpda_pwelch
ltpda_rand
ltpda_reformat_date
ltpda_ri2fq
ltpda_rms
retrieve
retrieve
retrieve
retrieve
retrieve
retrieve
retrieve
retrieve
retrieve
retrieve
retrieve
rp2iir
rstruct
rz2iir
save
save
save
save
save
save
save
save
save
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
wrapstring
xdoc2ao
xydata
xyzdata
zero

Generated on Tue 26-Feb-2008 10:52:32 by **m2html** © 2003

char

PURPOSE ^

CHAR overloads char() function for analysis objects.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
 CHAR overloads char() function for analysis objects.

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

 DESCRIPTION:     CHAR overloads char() function for analysis objects.

 CALL:            c = char(ao_in);

 FORMAT:          c = 'ao.name/ao.data.name class(ao.data) [size(ao.data.yaxis)]'

 POSSIBLE VALUES: ao_in  = [ao2 ao3]
                  ao_in  = ao_vector
                  ao_in  = ao_matrix

 VERSION:         $Id: char.m,v 1.10 2008/02/12 19:29:33 hewitson Exp $

 HISTORY: 15-02-07  M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.

This function is called by:

- abs ABS overloads the Absolute value operator for Analysis objects.
- acos ACOS overloads the acos operator for Analysis objects. Inverse cosine result in radians.
- ao AO analysis object class constructor.

- ao2m AO2M converts an analysis object to an '.m' file based on the history contained in the analysis object.
- asin ASIN overloads the asin operator for Analysis objects. Inverse sine result in radians.
- atan ATAN overloads the atan operator for Analysis objects. Inverse tangent result in radians.
- attachm ATTACHM attach the m file being executes to the analysis object.
- cat CAT concatonate AOs into a vector.
- char CHAR overloads char() function for analysis objects.
- complex COMPLEX overloads the complex operator 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.
- ctranspose CTRANSPOSE overloads the ' operator for Analysis Objects.
- demux DEMUX splits the input vector of AOs into a number of output AOs.
- det DET overloads the determinant function for Analysis objects.
- diag DIAG overloads the diagonal operator for Analysis Objects.
- display DISPLAY implement terminal display for analysis object.
- downsample DOWNSAMPLE AOs containing time-series data.
- 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.
- fft FFT overloads the fft operator for Analysis objects.
- filter FILTER overides the filter function for analysis objects.
- filtfilt FILTFILT overides the filtfilt function for analysis objects.
- find FIND particular samples that satisfy the input query and return a new AO.
- 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.
- 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 specified by the input parameter list.
- inv INV overloads the inverse function for Analysis Objects.
- join JOIN multiple AOs into a single AO.
- 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.
- 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.
- lt LT overloads < operator for analysis objects. Compare the y-axis values.
- mean MEAN overloads the mean operator for Analysis objects. Compute the mean value.
- median MEDIAN overloads the median operator for Analysis objects. Compute the median value.
- minus MINUS overloads - operator for analysis objects.
- mpower MPOWER implements power operator for analysis objects.
- mrdivide MRDIVIDE overloads / operator for analysis objects.
- mtimes MTIMES overloads * operator for analysis objects.
- mux MUX concatonate AOs into a vector.
- ne NE overloads the ~= operator for analysis objects.
- 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 two analysis objects.
- polyfit POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
- power POWER implements power operator for analysis objects.
- rdivide RDIVIDE overloads ./ operator for analysis objects.
- real REAL overloads the real operator for Analysis objects.
- resample RESAMPLE overloads resample function for AOs.
- save SAVE overloads save operator for analysis objects.
- select SELECT select particular samples from the input AOs and return new AOs with only those samples.
- sin SIN overloads the sin operator for Analysis objects. Sine of argument in radians.
- spectrogram SPECTROGRAM computes a spectrogram of the given ao/tsdata.
- split SPLIT split an analysis object into the specified segments.
- sqrt SQRT overloads the sqrt operator for Analysis objects. Square root.
- stairs STAIRS overloads the stairs function for Analysis Objects.
- std STD overloads the std operator for Analysis objects. Compute the standard deviation.
- subsasgn SUBSASGN define index assignment for analysis object properties.
- subsref SUBSREF Define field name indexing for analysis objects.
- sum SUM overloads the sum operator for Analysis objects. Compute the sum value.
- svd SVD overloads the determinant function for Analysis objects.
- tan TAN overloads the tan operator for Analysis objects. Tangent of argument in radians.
- times TIMES overloads .* 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.
- upsample UPSAMPLE overloads upsample function for AOs.
- validate VALIDATE checks that the input Analysis Object is reproducible and valid.
- var VAR overloads the var operator for Analysis objects. Compute the variance.

SOURCE CODE ^

```
function varargout = char(varargin)
% CHAR overloads char() function for analysis objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION:     CHAR overloads char() function for analysis objects.
%
% CALL:            c = char(ao_in);
%
% FORMAT:          c = 'ao.name/ao.data.name class(ao.data) [size(ao.data.
yaxis)]'
%
% POSSIBLE VALUES: ao_in  = [ao2 ao3]
%                  ao_in  = ao_vector
%                  ao_in  = ao_matrix
%
% VERSION:         $Id: char.m,v 1.10 2008/02/12 19:29:33 hewitson Exp $
%
% HISTORY: 15-02-07  M Hewitson
%               Creation
```

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

VERSION  = '$Id: char.m,v 1.10 2008/02/12 19:29:33 hewitson Exp $';
CATEGORY = 'Output';

%% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      varargout{1} = plist();
      return
    elseif strcmp(in, 'Version')
      varargout{1} = VERSION;
      return
    elseif strcmp(in, 'Category')
      varargout{1} = CATEGORY;
      return
    end
  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

[ao_set, pl, invars] = collect_inputs(varargin, in_names);

if ~isempty (pl)
  pl = combine(pl);
end

%% go through analysis objects
for j=1:numel(ao_set)

  a = ao_set(j);

  pstr = [];
  pstr = [pstr a.name];

  % examine data object
  d = a.data;

  % get data type
  if isempty(d)
    pstr = [pstr '  /  No data-object'];
  else
    pstr = [pstr '/' d.name '   ' class(d)];
```

```

    [x,y] = get_xy_values(a.data);

    xinfo = whos('y');
    pstr = [pstr '    ' sprintf('[%d %d]', xinfo.size(1), xinfo.size(2)) ];

  end
  % get data time

  pstr = strrep(pstr, '_', '\_');

  varargout{j} = pstr;

end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

find

PURPOSE ^

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

SYNOPSIS ^

function bo = 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

 VERSION:     $Id: find.m,v 1.13 2008/02/19 18:01:59 ingo Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = find(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = find(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = find(ao,'Category')

 HISTORY: 14-05-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- find FIND particular samples that satisfy the input query and return a new AO.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

- ao AO analysis object class constructor.
- ao2m AO2M converts an analysis object to an '.m' file based on the history contained in the analysis object.
- downsample DOWNSAMPLE AOs containing time-series data.
- filter FILTER overides the filter function for analysis objects.
- filtfilt FILTFILT overides the filtfilt function for analysis objects.
- find FIND particular samples that satisfy the input query and return a new AO.
- get GET get ao properties.
- hist HIST overloads the histogram function (hist) of MATLAB 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 specified by the input parameter list.
- iplot IPLOT provides an intelligent plotting tool for LTPDA.
- join JOIN multiple AOs into a single AO.
- mean MEAN overloads the mean operator for Analysis objects. Compute the mean value.
- median MEDIAN overloads the median operator for Analysis objects. Compute the median value.
- polyfit POLYFIT overloads polyfit() function of MATLAB for Analysis 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.
- spectrogram SPECTROGRAM computes a spectrogram of the given ao/tsdata.
- split SPLIT split an analysis object into the specified segments.
- std STD overloads the std operator for Analysis objects. Compute the standard deviation.
- sum SUM overloads the sum operator for Analysis objects. Compute the sum value.
- upsample UPSAMPLE overloads upsample function for AOs.
- validate VALIDATE checks that the input Analysis Object is reproducible and valid.
- var VAR overloads the var operator for Analysis objects. Compute the variance.

SUBFUNCTIONS ^

- function pl_default = getDefaultPL()

SOURCE CODE ^

```
function bo = find(varargin)
% 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
%
% VERSION:     $Id: find.m,v 1.13 2008/02/19 18:01:59 ingo Exp $
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = find(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = find(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = find(ao,'Category')
%
% HISTORY: 14-05-07 M Hewitson
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: find.m,v 1.13 2008/02/19 18:01:59 ingo Exp $';
ALGONAME = mfilename;
CATEGORY = 'Helper';
queries  = [];

%% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      bo = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      bo = VERSION;
      return
    elseif strcmp(in, 'Category')
      bo = CATEGORY;
      return
    end
  end
end

```

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

  if ischar(varargin{ii})
    queries = [queries ' & ' varargin{ii}];
  end
end

[as, ps, invars] = collect_inputs(varargin, in_names);

% Check inputs
if isempty(as)
  error('### Please input at least one AO.');
end
if isempty(ps) && isempty(queries)
  error('### Please specify queries to select directly or as a plist.');
end

pl = combine(ps);

% Get sample selection from plist
queries = [queries ' & ' find(pl, 'query')];

% We may have pre or trailing '&' now
queries = strtrim(queries);
if queries(1) == '&'
  queries = queries(2:end);
end
if queries(end) == '&'
  queries = queries(1:end-1);
end


% disp(sprintf('*** processing %s', queries));

% Loop over input AOs
bo = [];
for j=1:numel(as)

  a = as(j);
  d = a.data;

  [x,y] = get_xy_values(d);

  % special case for the cdata class.
  % The user can also use the query to 'vals'
  vals = y;

  cmd = sprintf('idx = find(%s);', queries);
  try
```

```
    eval(cmd);
  catch
    l_error = lasterror;
    error('%s\n\n### please use x or y for the queries.',l_error.message);
  end

  if isa(d, 'cdata')
    % Have each input of 'vals' a 'tags'
    if length(x) == length(y)
      d = set_xy_axis(d, x(idx), y(idx));
    else
      d = set_xy_axis(d, x, y(idx));
    end
  else
    d = set_xy_axis(d, x(idx), y(idx));
  end

  % Make output AO

  % create new output history
  queries = strtrim(queries);
  h = history(ALGONAME, VERSION, plist(param('query', queries)), a.hist);
  h = set(h, 'invars', invars(j));

  % make output analysis object
  b = ao(d, h);

  % set name
  b = setnh(b, 'name', sprintf('select(%s)', invars{j}));

  % Add to output array
  bo = [bo b];

end

% Reshape the ouput to the same size of the input
bo = reshape(bo, size(as));


%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist(param('query',  ''));



% END
```

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, N)
              b = hist(a, X)
              b = hist(a, pl)

 INPUTS:      a  - input analysis object(s)
              N  - number of bins
              X  - specify centers of bins
              pl - a parameter list

 OUTPUTS:     b  - xydata type analysis object(s) containing the
                   histogrammed data

 PARAMETERS:  'N'    - number of bins
              'X'    - set of bin centers

 VERSION:     $Id: hist.m,v 1.10 2008/02/12 08:31:45 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = hist(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = hist(ao,'Version')
```

```
 The following call returns a string that contains the routine category:

 >> category = hist(ao,'Category')

 HISTORY: 24-05-07 M Hewitson
             Creation

%%%
```

```
%               X  - specify centers of bins
%               pl - a parameter list
%
% OUTPUTS:      b  - xydata type analysis object(s) containing the
%                    histogrammed data
%
% PARAMETERS:   'N'     - number of bins
%               'X'     - set of bin centers
%
% VERSION:      $Id: hist.m,v 1.10 2008/02/12 08:31:45 mauro Exp $
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = hist(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = hist(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = hist(ao,'Category')
%
% HISTORY: 24-05-07 M Hewitson
%              Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ALGONAME = mfilename;
VERSION  = '$Id: hist.m,v 1.10 2008/02/12 08:31:45 mauro Exp $';
CATEGORY = 'Signal Processing';

%% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

%---------------- capture input variable names, AOs, and plists
Nbins  = -1;
```

```
Xbins  = [];

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

  % Collect Xbins and Nbins
  if isnumeric(varargin{ii})
    if length(varargin{ii}) > 1
      Xbins = varargin{ii};
    else
      Nbins = varargin{ii};
    end
  end
end

[as, ps, invars] = collect_inputs(varargin, in_names);

% produce one plist
if isa(ps, 'plist')
  pl = combine(ps, getDefaultPL());
else
  pl = getDefaultPL();
end

% If we have X from command input, override plist
if ~isempty(Xbins)
  pl = pset(pl, 'X', Xbins);
end

% If we have N from command input, override plist
if Nbins>0
  pl = pset(pl, 'N', Nbins);
end

% Get parameters
N = find(pl, 'N');
X = find(pl, 'X');

%---------------- Loop over input AOs

% Initialise output
bs = [];

% start looping
for j=1:numel(as)
  a = as(j);

  % get data
  [x, y] = get_xy_values(a.data);

```

```
  % Histogram this data
  if isempty(X)
    %disp(sprintf('** sorting data into %d bins', N));
    [n,x] = hist(y, N);
  else
    %disp(sprintf('** sorting data into %d bins', length(X)));
    [n,x] = hist(y, X);
  end

  %-------- Build output AOs

  % make a new xydata object
  xy = xydata(x, n);
  xy = set(xy, 'name', sprintf('hist(%s)', a.data.name));
  xy = set(xy, 'xunits', a.data.yunits);
  xy = set(xy, 'yunits', 'Count');

  % make a new history object
  h = history(ALGONAME, VERSION, pl, a.hist);
  h = set(h, 'invars', invars(j));

  % make output analysis object
  b = ao(xy, h);

  % name for this object
  b  = setnh(b, 'name', sprintf('hist(%s)', invars{j}));
  bs = [bs b];

end % end of AO loop

% Reshape the ouput to the same size of the input
bs = reshape(bs, size(as));


%---------------- Set outputs
varargout{1} = bs;

%----------------
%----------------
%----------------
%----------------

%-----------------------------------------------------------------------------

%% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('N', 10));
plo = append(plo, param('X', []));

```

```
% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(ao, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(ao, 'Version')

              The following call returns the category:
              >> ver = isfield(ao, 'Category')

 VERSION:     $Id: isfield.m,v 1.9 2008/02/12 09:29:24 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
              Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ao AO analysis object class constructor.

SOURCE CODE ^

```
function varargout = isfield(varargin)

% ISFIELD tests if the given field is one of the object properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: ISFIELD tests if the given field is one of the object
%              properties.
%
% CALL:        isfield(a, 'field');
%
%              The following call returns a parameter list object that
%              contains the default parameter values:
%              >> pl = isfield(ao, 'Params')
%
%              The following call returns the cvs version number:
%              >> ver = isfield(ao, 'Version')
%
%              The following call returns the category:
%              >> ver = isfield(ao, 'Category')
%
% VERSION:     $Id: isfield.m,v 1.9 2008/02/12 09:29:24 mauro Exp $
%
% HISTORY: 25-09-07 M Hewitson
%              Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: isfield.m,v 1.9 2008/02/12 09:29:24 mauro Exp $';
CATEGORY = 'Helper';

% 'Params', 'Version', 'Category' Call
if nargin == 2
  if isa(varargin{1}, 'ao') && strcmp(varargin{2}, 'Params')
    varargout{1} = plist();
    return
  elseif isa(varargin{1}, 'ao') && strcmp(varargin{2}, 'Version')
    varargout{1} = VERSION;
    return
  elseif isa(varargin{1}, 'ao') && strcmp(varargin{2}, 'Category')
    varargout{1} = CATEGORY;
    return
  end
end

as    = [];
field = '';
for j=1:nargin
  if isa(varargin{j}, 'ao')
```

```
    as = [as varargin{j}];
  end
  if ischar(varargin{j})
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

log10

PURPOSE ^

LOG10 overloads the log10 operator for Analysis objects. Common (base 10) logarithm.

SYNOPSIS ^

function ao_out = 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);

 POSSIBLE VALUES: ao_in  = [ao2 ao3]
                  ao_in  = ao_vector
                  ao_in  = ao_matrix

 PARAMETER LIST:  <key>           <value>             <description>
                          tsdata  fsdata  xydata
                 'xdata'   't'     'f'      'x'     compute the xdata
                 'xdata'   'x'     'xx'     'y'     compute the ydata

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = log10(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = log10(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = log10(ao,'Category')

 HISTORY: 23-05-2007 Diepholz
```

```
              Creation

%%%
```

```
% >> pl = log10(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = log10(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = log10(ao,'Category')
%
% HISTORY: 23-05-2007 Diepholz
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: log10.m,v 1.10 2008/02/12 09:29:24 mauro Exp $';
CATEGORY = 'Operator';

ao_out   = [];

%% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      ao_out = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      ao_out = VERSION;
      return
    elseif strcmp(in, 'Category')
      ao_out = CATEGORY;
      return
    end
  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

[ao_set, pl, invars] = collect_inputs(varargin, in_names);

if ~isempty (pl)
  pl = combine(pl);
end

%% go through analysis objects
for j=1:numel(ao_set)
  a = ao_set(j);

```

```
  [h, log10_data] = single_operation(a.data, 'log10', pl);

  %% Add the history from the ao object to the history
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  %% create a new analysis objects
  new_ao = a;
  new_ao = ao  (log10_data, h);
  new_ao = setnh(new_ao, 'name', sprintf('log10(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('xdata',  '')
                      param('ydata',  '')]);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

set

PURPOSE ^

SET set an analysis object property.

SYNOPSIS ^

function aos = set(varargin)

DESCRIPTION ^

```
 SET set an analysis object property.

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

 DESCRIPTION: SET set an analysis object property.

 CALL: ao = set(ao, 'name',       'name');
       ao = set(ao, 'provenance',  provenance-object);
       ao = set(ao, 'description','description');
       ao = set(ao, 'hist',        history-object);
       ao = set(ao, 'data',        data);
       ao = set(ao, 'mfile',       mfile);
       ao = set(ao, 'mfilename',  'mfilename');
       ao = set(ao, 'mdlfile',    'mdlfile');
       ao = set(ao, 'mdlfilename','mdlfilename');
       ao = set(ao, 'version',    'cvs-version');
       ao = set(ao, 'created',     time-object);
       ao = set(ao,  plist-object);          --> plist-object with key/value paies
       ao = set(ao, 'data-property', value); --> ao.data.?? = value;
       ao = set(ao, 'fs',          fs);      --> ao.data.fs = fs;
       ao = set(ao, 't0',          t0);      --> ao.data.t0 = t0;
       ao = set(ao, 'xunits',     'xunits'); --> ao.data.xunits = 'xunits';
       ao = set(ao, 'yunits',     'yunits'); --> ao.data.xunits = 'yunits';

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = set(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = set(ao,'Version')

 The following call returns a string that contains the routine category:
```

```
>> category = set(ao,'Category')

VERSION: $Id: set.m,v 1.32 2008/02/21 10:20:18 ingo Exp $

HISTORY: 30-01-07 M Hewitson
            Creation

%%%
```

- median MEDIAN overloads the median operator for Analysis objects. Compute the median value.
- minus MINUS overloads - operator for analysis objects.
- mpower MPOWER implements power operator for analysis objects.
- mrdivide MRDIVIDE overloads / operator for analysis objects.
- mtimes MTIMES overloads * operator for analysis objects.
- 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 two analysis objects.
- polyfit POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
- power POWER implements power operator for analysis objects.
- rdivide RDIVIDE overloads ./ operator for analysis objects.
- real REAL overloads the real operator for Analysis 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.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.
- sin SIN overloads the sin operator for Analysis objects. Sine of argument in radians.
- spectrogram SPECTROGRAM computes a spectrogram of the given ao/tsdata.
- split SPLIT split an analysis object into the specified segments.
- sqrt SQRT overloads the sqrt operator for Analysis objects. Square root.
- stairs STAIRS overloads the stairs function for Analysis Objects.
- std STD overloads the std operator for Analysis objects. Compute the standard deviation.
- sum SUM overloads the sum operator for Analysis objects. Compute the sum value.
- svd SVD overloads the determinant function for Analysis objects.
- tan TAN overloads the tan operator for Analysis objects. Tangent of argument in radians.
- times TIMES overloads .* operator for analysis objects.
- transpose TRANSPOSE overloads the .' operator for Analysis Objects.
- upsample UPSAMPLE overloads upsample function for AOs.
- var VAR overloads the var operator for Analysis objects. Compute the variance.

SOURCE CODE ^

```
function aos = set(varargin)
% SET set an analysis object property.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SET set an analysis object property.
%
% CALL: ao = set(ao, 'name',        'name');
%       ao = set(ao, 'provenance',  provenance-object);
%       ao = set(ao, 'description','description');
%       ao = set(ao, 'hist',        history-object);
%       ao = set(ao, 'data',        data);
%       ao = set(ao, 'mfile',       mfile);
%       ao = set(ao, 'mfilename',  'mfilename');
%       ao = set(ao, 'mdlfile',    'mdlfile');
```

```
%       ao = set(ao, 'mdlfilename','mdlfilename');
%       ao = set(ao, 'version',    'cvs-version');
%       ao = set(ao, 'created',     time-object);
%       ao = set(ao,  plist-object);          --> plist-object with key/value paies
%       ao = set(ao, 'data-property', value); --> ao.data.?? = value;
%       ao = set(ao, 'fs',          fs);      --> ao.data.fs = fs;
%       ao = set(ao, 't0',          t0);      --> ao.data.t0 = t0;
%       ao = set(ao, 'xunits',     'xunits'); --> ao.data.xunits = 'xunits';
%       ao = set(ao, 'yunits',     'yunits'); --> ao.data.xunits = 'yunits';
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = set(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = set(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = set(ao,'Category')
%
% VERSION: $Id: set.m,v 1.32 2008/02/21 10:20:18 ingo Exp $
%
% HISTORY: 30-01-07 M Hewitson
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% REMARK: This function don't use the generic set-function!!!

VERSION = '$Id: set.m,v 1.32 2008/02/21 10:20:18 ingo Exp $';
CATEGORY      = 'Helper';
DEFAULT_PLIST =  plist('name',        '',          ...
                       'data',        tsdata,      ...
                       'hist',        history,     ...
                       'provenance',  provenance,  ...
                       'description', '',          ...
                       'mfile',       {''},        ...
                       'mfilename',   '',          ...
                       'mdlfile',     '',          ...
                       'mdlfilename', '',          ...
                       'plist',       '',          ...
                       'version',     '',          ...
                       'created',     time,        ...
                       'fs',          [],          ...
                       't0',          time,        ...
                       'xunits',      '',          ...
                       'yunits',      '');

```

```
%%% Check if this is a special call: default parameter list, cvs-version,
if isa(varargin{1}, 'ao') && ischar(varargin{2})
  in = varargin{2};
  if strcmp(in, 'Params')
    aos = DEFAULT_PLIST;
    return
  elseif strcmp(in, 'Version')
    aos = VERSION;
    return
  elseif strcmp(in, 'Category')
    aos = CATEGORY;
    return
  end
end

aos       = [];
invars    = {};
pls       = plist();
propArgin = {};

%%% collect input ao's, plist's and ao variable names
for j=1:numel(varargin)

  if isa(varargin{j}, 'ao')
    aos = [aos varargin{j}];

    ao_name = inputname(j);
    if isempty(ao_name)
      ao_name = 'no_ao_name';
    end

    % Memorise the variable name of the corresponding analysis object.
    % If the ao is an array or vector add the index to the variable name
    if numel(varargin{j}) == 1
      invars{end+1} = ao_name;
    else
      for ii=1:numel(varargin{j})
        [I,J] = ind2sub(size(varargin{j}),ii);
        invars{end+1} = sprintf('%s(%d,%d)', ao_name, I, J);
      end
    end
  elseif isempty(propArgin) && isa(varargin{j}, 'plist')
    pls = [pls varargin{j}];
  else
    propArgin{end+1} = varargin{j};
  end
end

pls         = combine(pls);
ao_fields   = fieldnames(aos);
data_fields = '';
if ~isempty(aos(1).data)
```

```
  data_fields = fieldnames(aos(1).data);
end

%%% Exist a plist then add the key/value pairs to propArgin
if isempty(propArgin)
  for ii = 1:nparams(pls)
    %% The 'key' name of a plist/param is always stored with upper case
    %% characters. To set a ao property with this key we have to lower the
    %% 'key' characters.
    propArgin{end+1} = lower(pls.params(ii).key);
    propArgin{end+1} =       pls.params(ii).val;
  end
end

hist_pl = plist();

while length(propArgin) >= 2

  prop       = propArgin{1};
  val        = propArgin{2};
  propArgin = propArgin(3:end);

  %%% Normal case
  if ismember(prop, ao_fields)
    for jj = 1:numel(aos)
      aos(jj).(prop) = val;
    end

  %%% Special case: Is prop a property of the data class then set the data class.
  elseif ismember(prop, data_fields)
    for jj = 1:numel(aos)
      aos(jj).data = set(aos(jj).data, prop, val);
    end

  else
    error('### ''%s'' is not a valid %s-object property.', prop, class(aos));
  end

  hist_pl = append(hist_pl, prop, val);

end

for jj = 1:numel(aos)
  h = history('set', VERSION, hist_pl);
  h = set(h, 'inhists', aos(jj).hist);
  h = set(h, 'invars',  cellstr(invars{jj}));

  aos(jj).hist = h;
end

if (nargout == 0)
```

```
  display(aos);
  error('\n### No output variable!\n### Please use: %s = set(%s, ''key'', ''value'');', inputname(1), inputname(1));
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

setnh

PURPOSE ^

SET set an analysis object property without recording the history.

SYNOPSIS ^

function aos = setnh(varargin)

DESCRIPTION ^

```
 SET set an analysis object property without recording the history.

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

 DESCRIPTION: SET set an analysis object property without recording the history.

 CALL: ao = setnh(ao, 'name',        'name');
       ao = setnh(ao, 'provenance',  provenance-object);
       ao = setnh(ao, 'description','description');
       ao = setnh(ao, 'hist',        history-object);
       ao = setnh(ao, 'data',        data);
       ao = setnh(ao, 'mfile',       mfile);
       ao = setnh(ao, 'mfilename',  'mfilename');
       ao = setnh(ao, 'mdlfile',    'mdlfile');
       ao = setnh(ao, 'mdlfilename','mdlfilename');
       ao = setnh(ao, 'version',    'cvs-version');
       ao = setnh(ao, 'created',     time-object);
       ao = setnh(ao,  plist-object);         --> plist-object with key/value paies
       ao = setnh(ao, 'fs',          fs);     --> ao.data.fs = fs;
       ao = setnh(ao, 't0',          t0);     --> ao.data.t0 = t0;
       ao = setnh(ao, 'xunits',     'xunits'); --> ao.data.xunits = 'xunits';
       ao = setnh(ao, 'yunits',     'yunits'); --> ao.data.xunits = 'yunits';

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = setnh(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = setnh(ao,'Version')

 The following call returns a string that contains the routine category:
```

```
>> category = setnh(ao,'Category')

VERSION: $Id: setnh.m,v 1.9 2008/02/21 10:20:18 ingo Exp $

HISTORY: 30-01-07 M Hewitson
            Creation

%%%
```

- mrdivide MRDIVIDE overloads / operator for analysis objects.
- mtimes MTIMES overloads * 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 two analysis objects.
- polyfit POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
- power POWER implements power operator for analysis objects.
- rdivide RDIVIDE overloads ./ operator for analysis objects.
- real REAL overloads the real operator for Analysis 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.
- sin SIN overloads the sin operator for Analysis objects. Sine of argument in radians.
- spectrogram SPECTROGRAM computes a spectrogram of the given ao/tsdata.
- sqrt SQRT overloads the sqrt operator for Analysis objects. Square root.
- std STD overloads the std operator for Analysis objects. Compute the standard deviation.
- sum SUM overloads the sum operator for Analysis objects. Compute the sum value.
- svd SVD overloads the determinant function for Analysis objects.
- tan TAN overloads the tan operator for Analysis objects. Tangent of argument in radians.
- times TIMES overloads .* operator for analysis objects.
- transpose TRANSPOSE overloads the .' operator for Analysis Objects.
- upsample UPSAMPLE overloads upsample function for AOs.
- var VAR overloads the var operator for Analysis objects. Compute the variance.

SOURCE CODE ^

```
function aos = setnh(varargin)
% SET set an analysis object property without recording the history.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SET set an analysis object property without recording the history.
%
% CALL: ao = setnh(ao, 'name',       'name');
%       ao = setnh(ao, 'provenance',  provenance-object);
%       ao = setnh(ao, 'description','description');
%       ao = setnh(ao, 'hist',        history-object);
%       ao = setnh(ao, 'data',        data);
%       ao = setnh(ao, 'mfile',       mfile);
%       ao = setnh(ao, 'mfilename',  'mfilename');
%       ao = setnh(ao, 'mdlfile',    'mdlfile');
%       ao = setnh(ao, 'mdlfilename','mdlfilename');
%       ao = setnh(ao, 'version',    'cvs-version');
%       ao = setnh(ao, 'created',     time-object);
%       ao = setnh(ao,  plist-object);         --> plist-object with key/value paies
%       ao = setnh(ao, 'fs',          fs);     --> ao.data.fs = fs;
%       ao = setnh(ao, 't0',          t0);     --> ao.data.t0 = t0;
%       ao = setnh(ao, 'xunits',     'xunits'); --> ao.data.xunits = 'xunits';
```

```
%       ao = setnh(ao, 'yunits',     'yunits'); --> ao.data.xunits = 'yunits';
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = setnh(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = setnh(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = setnh(ao,'Category')
%
% VERSION: $Id: setnh.m,v 1.9 2008/02/21 10:20:18 ingo Exp $
%
% HISTORY: 30-01-07 M Hewitson
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% REMARK: This function don't use the generic set-function!!!

VERSION = '$Id: setnh.m,v 1.9 2008/02/21 10:20:18 ingo Exp $';
CATEGORY      = 'Helper';
DEFAULT_PLIST =  plist('name',        '',          ...
                       'data',        tsdata,      ...
                       'hist',        history,     ...
                       'provenance',  provenance,  ...
                       'description', '',          ...
                       'mfile',       {''},        ...
                       'mfilename',   '',          ...
                       'mdlfile',     '',          ...
                       'mdlfilename', '',          ...
                       'plist',       '',          ...
                       'version',     '',          ...
                       'created',     time,        ...
                       'fs',          [],          ...
                       't0',          time,        ...
                       'xunits',      '',          ...
                       'yunits',      '');

%%% Check if this is a special call: default parameter list, cvs-version,
if isa(varargin{1}, 'ao') && ischar(varargin{2})
  in = varargin{2};
  if strcmp(in, 'Params')
    aos = DEFAULT_PLIST;
    return
  elseif strcmp(in, 'Version')
    aos = VERSION;
    return
```

```
  elseif strcmp(in, 'Category')
    aos = CATEGORY;
    return
  end
end

aos       = [];
invars    = {};
pls       = plist();
propArgin = {};

%%% collect input ao's, plist's and ao variable names
for j=1:numel(varargin)

  if isa(varargin{j}, 'ao')
    aos = [aos varargin{j}];

  elseif isempty(propArgin) && isa(varargin{j}, 'plist')
    pls = [pls varargin{j}];

  else
    propArgin{end+1} = varargin{j};
  end
end

pls         = combine(pls);
ao_fields   = fieldnames(aos);
data_fields = '';
if ~isempty(aos(1).data)
  data_fields = fieldnames(aos(1).data);
end

%%% Exist a plist then add the key/value pairs to propArgin
if isempty(propArgin)
  for ii = 1:nparams(pls)
    %% The 'key' name of a plist/param is always stored with upper case
    %% characters. To set a ao property with this key we have to lower the
    %% 'key' characters.
    propArgin{end+1} = lower(pls.params(ii).key);
    propArgin{end+1} =       pls.params(ii).val;
  end
end

while length(propArgin) >= 2

  prop       = propArgin{1};
  val        = propArgin{2};
  propArgin = propArgin(3:end);

  %%% Normal case
  if ismember(prop, ao_fields)
    for jj = 1:numel(aos)
```

```
      aos(jj).(prop) = val;
    end

  %%% Special case: Is prop a property of the data class then set the data class.
  elseif ismember(prop, data_fields)
    for jj = 1:numel(aos)
      aos(jj).data = set(aos(jj).data, prop, val);
    end

  else
    error('### ''%s'' is not a valid %s-object property.', prop, class(aos));
  end

end

if (nargout == 0)
  display(aos);
  error('\n### No output variable!\n### Please use: %s = set(%s, ''key'', ''value'');', inputname(1), inputname(1));
end

```

abs

PURPOSE ^

ABS overloads the Absolute value operator for Analysis objects.

SYNOPSIS ^

function ao_out = abs(varargin)

DESCRIPTION ^

```
 ABS overloads the Absolute value operator for Analysis objects.

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

 DESCRIPTION: ABS overloads the Absolute value operator for Analysis objects.

 CALL: ao_out = abs(ao_in);
       ao_out = abs(ao_in, pl);

 POSSIBLE VALUES: ao_in  = [ao2 ao3]
                  ao_in  = ao_vector
                  ao_in  = ao_matrix

 PARAMETER LIST:  <key>           <value>             <description>
                          tsdata  fsdata  xydata
                 'xdata'   't'     'f'      'x'     compute the xdata
                 'ydata'   'x'     'xx'     'y'     compute the ydata

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = abs(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = abs(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = abs(ao,'Category')

 HISTORY: 12-03-07 M Hewitson
             Creation
```

```
%%%
```

```
% The following call returns a string that contains the routine category:
%
% >> category = abs(ao,'Category')
%
% HISTORY: 12-03-07 M Hewitson
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: abs.m,v 1.13 2008/02/12 08:31:45 mauro Exp $';
CATEGORY = 'Operator';

ao_out = [];

%% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      ao_out = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      ao_out = VERSION;
      return
    elseif strcmp(in, 'Category')
      ao_out = CATEGORY;
      return
    end
  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

[aos, pls, invars] = collect_inputs(varargin, in_names);

if ~isempty (pls)
  pls = combine(pls);
end

%% go through analysis objects
for j=1:numel(aos)
  a = aos(j);

  [h, abs_data] = single_operation(a.data, 'abs', pls);

  %% Set the x, y-units to their old values (without abs())
  abs_data = set(abs_data, 'xunits', a.data.xunits);
  abs_data = set(abs_data, 'yunits', a.data.yunits);
```

```
  %% It is nicer with the absolute signs | | than abs ()
  abs_data = set(abs_data, 'name', sprintf ('| %s |',a.data.name));

  %% Add the history from the ao object to the history
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  %% create a new analysis objects
  new_ao = ao  (abs_data, h);
  new_ao = setnh(new_ao, 'name', sprintf('|%s|',invars{j}) );

  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(aos));


%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('xdata',  '')
                      param('ydata',  '')]);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

acos

PURPOSE ^

ACOS overloads the acos operator for Analysis objects. Inverse cosine result in radians.

SYNOPSIS ^

function ao_out = acos (varargin)

DESCRIPTION ^

```
 ACOS overloads the acos operator for Analysis objects. Inverse cosine result in
radians.

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

 DESCRIPTION: ACOS overloads the acos operator for Analysis objects.
              Inverse cosine result in radians.
              ACOS(ao) is the cosine of the elements of ao.data.

 CALL: ao_out = acos(ao_in);
       ao_out = acos(ao_in, pl);
       ao_out = acos(ao1, pl1, ao_vector, ao_matrix, pl2);

 POSSIBLE VALUES: ao_in  = [ao2 ao3]
                  ao_in  = ao_vector
                  ao_in  = ao_matrix

 PARAMETER LIST:  <key>           <value>             <description>
                         tsdata  fsdata  xydata
                 'xdata'   't'     'f'      'x'     compute the xdata
                 'ydata'   'x'     'xx'     'y'     compute the ydata

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = acos(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = acos(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = acos(ao,'Category')
```

```
 HISTORY: 07-05-2007 Diepholz
             Creation

%%%
```

```
% >> pl = acos(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = acos(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = acos(ao,'Category')
%
% HISTORY: 07-05-2007 Diepholz
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: acos.m,v 1.11 2008/02/12 08:31:45 mauro Exp $';
CATEGORY = 'Trigonometry';
ao_out   = [];

%% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      ao_out = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      ao_out = VERSION;
      return
    elseif strcmp(in, 'Category')
      ao_out = CATEGORY;
      return
    end
  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

[ao_set, pl, invars] = collect_inputs(varargin, in_names);

if ~isempty (pl)
  pl = combine(pl);
end

%% go through analysis objects
for j=1:numel(ao_set)
  a = ao_set(j);

```

```
  [h, acos_data] = single_operation(a.data, 'acos', pl);

  %% Add the history from the ao object to the history
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  %% create a new analysis objects
  new_ao = a;
  new_ao = ao  (acos_data, h);
  new_ao = setnh(new_ao, 'name', sprintf('acos(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('xdata',  '')
                      param('ydata',  '')]);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

asin

PURPOSE ^

ASIN overloads the asin operator for Analysis objects. Inverse sine result in radians.

SYNOPSIS ^

function ao_out = asin (varargin)

DESCRIPTION ^

```
 ASIN overloads the asin operator for Analysis objects. Inverse sine result in
radians.

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

 DESCRIPTION: ASIN overloads the asin operator for Analysis objects.
              Inverse sine result in radians.
              ASIN(ao) is the arcsine of the elements of ao.data.

 CALL: ao_out = asin(ao_in);
       ao_out = asin(ao_in, pl);
       ao_out = asin(ao1, pl1, ao_vector, ao_matrix, pl2);

 POSSIBLE VALUES: ao_in  = [ao2 ao3]
                  ao_in  = ao_vector
                  ao_in  = ao_matrix

 PARAMETER LIST:  <key>           <value>             <description>
                          tsdata  fsdata  xydata
                 'xdata'   't'     'f'      'x'     compute the xdata
                 'ydata'   'x'     'xx'     'y'     compute the ydata

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = asin(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = asin(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = asin(ao,'Category')
```

```
 HISTORY: 07-05-2007 Diepholz
             Creation

%%%
```

```
% The following call returns a string that contains the routine CVS version:
%
% >> version = asin(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = asin(ao,'Category')
%
% HISTORY: 07-05-2007 Diepholz
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: asin.m,v 1.10 2008/02/12 08:31:45 mauro Exp $';
CATEGORY = 'Trigonometry';
ao_out   = [];

%% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      ao_out = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      ao_out = VERSION;
      return
    elseif strcmp(in, 'Category')
      ao_out = CATEGORY;
      return
    end
  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

[ao_set, pl, invars] = collect_inputs(varargin, in_names);

if ~isempty (pl)
  pl = combine(pl);
end

%% go through analysis objects
for j=1:numel(ao_set)
  a = ao_set(j);

  [h, asin_data] = single_operation(a.data, 'asin', pl);

  %% Add the history from the ao object to the history
```

```
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  %% create a new analysis objects
  new_ao = ao  (asin_data, h);
  new_ao = setnh (new_ao, 'name', sprintf('asin(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('xdata',  '')
                      param('ydata',  '')]);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

atan

PURPOSE ^

ATAN overloads the atan operator for Analysis objects. Inverse tangent result in radians.

SYNOPSIS ^

function ao_out = atan (varargin)

DESCRIPTION ^

```
 ATAN overloads the atan operator for Analysis objects. Inverse tangent result in
radians.

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

 DESCRIPTION: ATAN overloads the atan operator for Analysis objects.
              Inverse tangent result in radians.
              ATAN(ao) is the arctangent of the elements of ao.data.

 CALL: ao_out = atan(ao_in);
       ao_out = atan(ao_in, pl);
       ao_out = atan(ao1, pl1, ao_vector, ao_matrix, pl2);

 POSSIBLE VALUES: ao_in  = [ao2 ao3]
                  ao_in  = ao_vector
                  ao_in  = ao_matrix

 PARAMETER LIST:  <key>           <value>             <description>
                          tsdata  fsdata  xydata
                 'xdata'   't'     'f'      'x'     compute the xdata
                 'ydata'   'x'     'xx'     'y'     compute the ydata

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = atan(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = atan(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = atan(ao,'Category')
```

```
 HISTORY: 07-05-2007 Diepholz
             Creation

%%%
```

```
% The following call returns a string that contains the routine CVS version:
%
% >> version = atan(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = atan(ao,'Category')
%
% HISTORY: 07-05-2007 Diepholz
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: atan.m,v 1.10 2008/02/12 08:31:45 mauro Exp $';
CATEGORY = 'Trigonometry';
ao_out   = [];

%% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      ao_out = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      ao_out = VERSION;
      return
    elseif strcmp(in, 'Category')
      ao_out = CATEGORY;
      return
    end
  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

[ao_set, pl, invars] = collect_inputs(varargin, in_names);

if ~isempty (pl)
  pl = combine(pl);
end

%% go through analysis objects
for j=1:numel(ao_set)
  a = ao_set(j);

  [h, atan_data] = single_operation(a.data, 'atan', pl);

  %% Add the history from the ao object to the history
```

```
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  %% create a new analysis objects
  new_ao = ao  (atan_data, h);
  new_ao = setnh (new_ao, 'name', sprintf('atan(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('xdata',  '')
                      param('ydata',  '')]);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

complex

PURPOSE ^

COMPLEX overloads the complex operator for Analysis objects.

SYNOPSIS ^

function ao_out = complex(a1, a2)

DESCRIPTION ^

```
 COMPLEX overloads the complex operator for Analysis objects.

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

 DESCRIPTION: COMPLEX overloads the complex operator for Analysis objects.
              A3 = COMPLEX(A1,A2) returns the complex result A + Bi, where A and B
              are identically sized real arrays.

 CALL:        ao_out = complex(a1, a2);

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = imag(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = imag(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = imag(ao,'Category')

 VERSION:     $Id: complex.m,v 1.8 2008/02/12 08:31:45 mauro Exp $

 HISTORY:     20-08-2007 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- complex COMPLEX overloads the complex operator for Analysis objects.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

- complex COMPLEX overloads the complex operator for Analysis objects.

SOURCE CODE ^

```
function ao_out = complex(a1, a2)
% COMPLEX overloads the complex operator for Analysis objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: COMPLEX overloads the complex operator for Analysis objects.
%              A3 = COMPLEX(A1,A2) returns the complex result A + Bi, where A and B
%              are identically sized real arrays.
%
% CALL:        ao_out = complex(a1, a2);
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = imag(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = imag(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = imag(ao,'Category')
%
% VERSION:     $Id: complex.m,v 1.8 2008/02/12 08:31:45 mauro Exp $
%
% HISTORY:     20-08-2007 Diepholz
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ALGONAME = mfilename;
VERSION  = '$Id: complex.m,v 1.8 2008/02/12 08:31:45 mauro Exp $';
CATEGORY = 'Operator';

% Check if this is a call for parameters
if nargin == 2
```

```
  if isa(a1, 'ao') && ischar(a2)
    in = char(a2);
    if strcmp(in, 'Params')
      ao_out = plist();
      return
    elseif strcmp(in, 'Version')
      ao_out = VERSION;
      return
    elseif strcmp(in, 'Category')
      ao_out = CATEGORY;
      return
    end
  end
end

% Check for the same data.
if ~strcmp(class(a1.data), class(a2.data))
  error ('### The data class must be the same. (%s <-> %s)', ...
          class(a1.data), class(a2.data));
end

% Check for the same sampe rate
fields = fieldnames(a1.data);
if ismember('fs', fields)
  if a1.data.fs ~= a2.data.fs
    error('### The sample rate is not the same. Please resaple one of the AO''s');
  end
end

% Check the length of the AO's
[x1,y1] = get_xy_values(a1.data);
[x2,y2] = get_xy_values(a2.data);
if length(x1) ~= length(x2) || length(y1) ~= length(y2)
  error ('### The length of the data vectors must be the same.')
end

complex_x = x1;
complex_y = complex(y1, y2);

cmd = sprintf('data = %s(complex_x, complex_y);', class(a1.data));
eval(cmd);

data = set(data, 'name',  'complex');
data = set(data, 'xunits', a1.data.xunits);
data = set(data, 'yunits', a1.data.yunits);

h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
h = set(h, 'invars', {inputname(1), inputname(2)});

ao_out = ao(data, h);
```

```
ao_out = setnh(ao_out, 'name', sprintf('complex(%s, %s)', inputname(1),
inputname(2)));
```

Generated on Tue 26-Feb-2008 10:52:52 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.

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

 DESCRIPTION: CONJ overloads the conjugate operator for Analysis objects.

 CALL:        a_conj = conj

 VERSION:     $Id: conj.m,v 1.14 2008/02/12 08:31:45 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = conj(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = conj(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = conj(ao,'Category')

 HISTORY: 12-03-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.

- char CHAR overloads char() function for analysis objects.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = conj(varargin)
% CONJ overloads the conjugate operator for Analysis objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: CONJ overloads the conjugate operator for Analysis objects.
%
% CALL:        a_conj = conj
%
% VERSION:     $Id: conj.m,v 1.14 2008/02/12 08:31:45 mauro Exp $
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = conj(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = conj(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = conj(ao,'Category')
%
% HISTORY: 12-03-07 M Hewitson
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Check if this is a call for parameters

VERSION  = '$Id: conj.m,v 1.14 2008/02/12 08:31:45 mauro Exp $';
CATEGORY = 'Operator';
bs      = [];

if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
```

```
    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
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, ps, invars] = collect_inputs(varargin, in_names);

% check plist
if isempty(ps)
  pl = getDefaultPL();
else
  pl = combine(ps, getDefaultPL);
end

%% go through analysis objects
for j=1:numel(as)
  a = as(j);

  [h, conj_data] = single_operation(a.data, 'conj', pl);

  %% Set the x, y-units to their old values (without abs())
  conj_data = set(conj_data, 'xunits', a.data.xunits);
  conj_data = set(conj_data, 'yunits', a.data.yunits);
  conj_data = set(conj_data, 'name', sprintf ('conj(%s)',a.data.name));

  %% Add the history from the ao object to the history
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  %% create a new analysis objects
  new_ao = ao  (conj_data, h);
  new_ao = setnh(new_ao, 'name', sprintf('conj(%s)',invars{j}) );

  % add to output
  bs = [bs new_ao];
```

```
end

% Reshape the ouput to the same size of the input
varargout{1} = reshape(bs, size(as));

%% Get default params
function plo = getDefaultPL()
plo = plist();

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

cos

PURPOSE ^

COS overloads the cos operator for Analysis objects. Cosine of argument in radians.

SYNOPSIS ^

function ao_out = 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

 PARAMETER LIST:  <key>           <value>             <description>
                          tsdata  fsdata  xydata
                 'xdata'   't'     'f'      'x'     compute the xdata
                 'ydata'   'x'     'xx'     'y'     compute the ydata

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = cos(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = cos(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = cos(ao,'Category')
```

```
 HISTORY: 07-05-2007 Diepholz
             Creation

%%%
```

```
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = cos(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = cos(ao,'Category')
%
% HISTORY: 07-05-2007 Diepholz
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: cos.m,v 1.10 2008/02/12 08:31:45 mauro Exp $';
CATEGORY = 'Trigonometry';
ao_out   = [];

%% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      ao_out = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      ao_out = VERSION;
      return
    elseif strcmp(in, 'Category')
      ao_out = CATEGORY;
      return
    end
  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

[ao_set, pl, invars] = collect_inputs(varargin, in_names);

if ~isempty (pl)
  pl = combine(pl);
end

%% go through analysis objects
for j=1:numel(ao_set)
  a = ao_set(j);

  [h, cos_data] = single_operation(a.data, 'cos', pl);
```

```

  %% Add the history from the ao object to the history
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  %% create a new analysis objects
  new_ao = a;
  new_ao = ao  (cos_data, h);
  new_ao = setnh (new_ao, 'name', sprintf('cos(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('xdata',  '')
                      param('ydata',  '')]);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ctranspose

PURPOSE ^

CTRANSPOSE overloads the ' operator for Analysis Objects.

SYNOPSIS ^

function varargout = ctranspose(varargin)

DESCRIPTION ^

```
 CTRANSPOSE overloads the ' operator for Analysis Objects.

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

 DESCRIPTION: CTRANSPOSE overloads the ' operator for Analysis Objects.

 CALL:        a = a1'    % only with data = cdata

 VERSION:     $Id: ctranspose.m,v 1.15 2008/02/12 08:31:45 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = ctranspose(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = ctranspose(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = ctranspose(ao,'Category')

 HISTORY: 08-05-07 A Monsky
            Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.

- char CHAR overloads char() function for analysis objects.
- get GET get ao properties.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = ctranspose(varargin)
% CTRANSPOSE overloads the ' operator for Analysis Objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: CTRANSPOSE overloads the ' operator for Analysis Objects.
%
% CALL:        a = a1'    % only with data = cdata
%
% VERSION:     $Id: ctranspose.m,v 1.15 2008/02/12 08:31:45 mauro Exp $
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = ctranspose(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = ctranspose(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = ctranspose(ao,'Category')
%
% HISTORY: 08-05-07 A Monsky
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: ctranspose.m,v 1.15 2008/02/12 08:31:45 mauro Exp $';
CATEGORY = 'Operator';
bs       = [];

%% Check if this is a call for parameters

if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
```

```
    in = char(varargin{2});
    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
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, ps, invars] = collect_inputs(varargin, in_names);

% check plist
if isempty(ps)
  pl = getDefaultPL();
else
  pl = combine(ps, getDefaultPL);
end

%% go through analysis objects
for j=1:numel(as)
  a = as(j);

  d = get(a, 'data');
  dinfo = whos('d');
  % Which data type do we have
  dtype = dinfo.class;

  switch dtype
    case 'cdata'
%       disp('* CTRANSPOSE of cdata object');
      % make a new cdata object
      [h, c] = single_operation(d, 'ctranspose', pl);

      % make output analysis object
      h = set(h, 'inhists', [a.hist]);

      %% Set the var_name to the history
      h = set(h, 'invars', cellstr(invars{j}));

      b = ao(c, h);
```

```
      b = setnh(b, 'name',  sprintf('ctranspose(%s)', char(invars{j})));

    case {'tsdata','fsdata','xydata'}
      error('### this function works for cdata type AO only')
    otherwise
      error('### unknown data type.')

  end

  % 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;

%% Get default params
function plo = getDefaultPL()

plo = plist();

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

det

PURPOSE ^

DET overloads the determinant function for Analysis objects.

SYNOPSIS ^

function varargout = det(varargin)

DESCRIPTION ^

```
 DET overloads the determinant function for Analysis objects.

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

 DESCRIPTION: DET overloads the determinant function for Analysis objects.

 CALL:        a = det(a1)   % only with data = cdata

 VERSION:     $Id: det.m,v 1.17 2008/02/12 08:31:45 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = det(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = det(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = det(ao,'Category')

 HISTORY: 08-05-07 A Monsky
            Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.

- char CHAR overloads char() function for analysis objects.
- get GET get ao properties.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = det(varargin)
% DET overloads the determinant function for Analysis objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: DET overloads the determinant function for Analysis objects.
%
% CALL:        a = det(a1)   % only with data = cdata
%
% VERSION:     $Id: det.m,v 1.17 2008/02/12 08:31:45 mauro Exp $
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = det(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = det(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = det(ao,'Category')
%
% HISTORY: 08-05-07 A Monsky
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: det.m,v 1.17 2008/02/12 08:31:45 mauro Exp $';
CATEGORY = 'Operator';
bs       = [];

%% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
```

```
    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
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, ps, invars] = collect_inputs(varargin, in_names);

% check plist
if isempty(ps)
  pl = getDefaultPL();
else
  pl = combine(ps, getDefaultPL);
end

%% go through analysis objects
for j=1:numel(as)
  a = as(j);

  d = get(a, 'data');
  dinfo = whos('d');
  % Which data type do we have
  dtype = dinfo.class;

  switch dtype
    case 'cdata'
      % make a new cdata object

      if size(d.y,1) == size(d.y,2)
        [h, c] = single_operation(d, 'det', pl);
      else
        error('### The value (y) of data must be a square matrix.')
      end

      % make output analysis object
      h = set(h, 'inhists', [a.hist]);

      %% Set the var_name to the history
```

```
      h = set(h, 'invars', cellstr(invars{j}));

      b = ao(c, h);
      b = setnh(b, 'name',  sprintf('det(%s)', char(invars{j})));
    case {'tsdata','fsdata','xydata'}
      error('### this function works for cdata type AO only')
    otherwise
      error('### unknown data type.')
  end

  % add to output
  bs = [bs b];
end

bs = reshape(bs, size(as));
varargout{1} = bs;

%% Get default params
function plo = getDefaultPL()

plo = plist();


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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:  'dim' - The value of the dimension must be zero or a positiv
                      number.
                      e.g. pl = plist('dim', 1); This parameter list computes
                      the first 'sub' diagonal.

 VERSION:     $Id: diag.m,v 1.19 2008/02/12 08:31:45 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = diag(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = diag(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = diag(ao,'Category')
```

```
 HISTORY: 08-05-07 A Monsky
             Creation

%%%
```

```
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = diag(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = diag(ao,'Category')
%
% HISTORY: 08-05-07 A Monsky
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: diag.m,v 1.19 2008/02/12 08:31:45 mauro Exp $';
CATEGORY = 'Operator';

%% Check if this is a call for parameters

if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

%% capture input variable names
invars = {};

as = [];
bs = [];
pl = [];

for j=1:nargin
  if isa(varargin{j}, 'ao')
    as = [as varargin{j}];

    % Memorise the variable name of the corresponding analysis object.
    % If the ao is an array or vector add the index to the variable name
    if numel(varargin{j}) == 1
      invars{end+1} = inputname(j);
    else
      for ii=1:numel(varargin{j})
```

```
        [I,J] = ind2sub(size(varargin{j}),ii);
        invars{end+1} = sprintf('%s(%d,%d)', inputname(j), I, J);
      end
    end
  end
  if isa(varargin{j}, 'plist')
    pl = [pl varargin{j}];
  end
end

if ~isempty (pl)
  pl = combine(pl);
end


%% go through analysis objects
for j=1:numel(as)
  a = as(j);

  d = get(a, 'data');
  dinfo = whos('d');

  % Which data type do we have
  dtype = dinfo.class;
  switch dtype

    case 'cdata'
      %disp('* DIAGONAL of cdata object');
      % make a new cdata object
      [h, c] = single_operation(d, 'diag',pl);

      % make output analysis object
      h = set(h, 'inhists', [a.hist]);

      h = set(h, 'invars', cellstr(invars{j}));

      b = ao(c, h);

      b = setnh(b, 'name',  sprintf('diag(%s)', invars{j}));

    case {'tsdata','fsdata','xydata'}
      error('### this function works for cdata type AO only')
    otherwise
      error('### unknown data type.')
  end

  % 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;

%% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, 'option', 0);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY implement terminal display for analysis object.

SYNOPSIS ^

function txt = display(ao, varargin)

DESCRIPTION ^

```
 DISPLAY implement terminal display for analysis object.

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

 DESCRIPTION: DISPLAY implement terminal display for analysis object.

 CALL:        txt = display(ao)
              ao                     % without a semicolon

 VERSION:     $Id: display.m,v 1.18 2008/02/23 18:07:18 hewitson Exp $

 HISTORY:     31-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- get GET get ao properties.
- hist HIST overloads the histogram function (hist) of MATLAB for Analysis Objects.

This function is called by:

- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

SOURCE CODE ^

```
function txt = display(ao, varargin)
% DISPLAY implement terminal display for analysis object.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: DISPLAY implement terminal display for analysis object.
%
% CALL:        txt = display(ao)
%              ao                     % without a semicolon
%
% VERSION:     $Id: display.m,v 1.18 2008/02/23 18:07:18 hewitson Exp $
%
% HISTORY:     31-01-2007 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: display.m,v 1.18 2008/02/23 18:07:18 hewitson Exp $';
CATEGORY = 'Output';

%% Check if this is a call for parameters
if nargin == 2
  if isa(ao, 'ao') && ischar(varargin{1})
    in = char(varargin{1});
    if strcmp(in, 'Params')
      txt = plist();
      return
    elseif strcmp(in, 'Version')
      txt = VERSION;
      return
    elseif strcmp(in, 'Category')
      txt = CATEGORY;
      return
    end
  end
end

txt = {};

for i=1:numel(ao)
  a = ao(i);
  data   = get(a, 'data');
  dinfo  = whos('data');
  hist   = get(a, 'hist');
  hinfo  = whos('hist');

  if nargin ~= length(ao)
    banner_start = sprintf('----------- ao %02d -----------', i);
  else
    banner_start = sprintf('----------- ao: %s -----------', inputname(i));
  end
```

```

  txt{end+1} = banner_start;
  txt{end+1} = ' ';
  txt{end+1} = sprintf('       name:  %s', get(a, 'name'));
  txt{end+1} = sprintf(' provenance:  %s', char(get(a, 'provenance')));
  txt{end+1} = sprintf('description:  %s', char(get(a, 'description')));
  if isa(data, 'cell')
    txt{end+1} = sprintf('       data:  None');
  else
    mi = min(5,length(data.y));
    dtxt = sprintf('       data:  %s / %s [%dx%d] |', ...
                   dinfo.class, get(data, 'name'), ...
                   size(data.y,1), ...
                   size(data.y,2));
    for j=1:mi
      if length(data.x) == length(data.y)
        dtxt = [dtxt sprintf(' (%g,%g)', data.x(j), data.y(j))];
      else
        dtxt = [dtxt sprintf(' %g', data.y(j))];
      end
    end
    if mi < length(data.y)
      dtxt = [dtxt ' ...'];
    end
    txt{end+1} = dtxt;
  end
  if isa(hist, 'cell')
    txt{end+1} = sprintf('       hist:  None');
  else
    txt{end+1} = sprintf('       hist:  %s / %s / %s', hinfo.class, get(hist, 'name'), get(hist, 'version'));
  end
  txt{end+1} = sprintf('  mfilename:  %s', a.mfilename);
  txt{end+1} = sprintf('mdlfilename:  %s', a.mdlfilename);

  banner_end(1:length(banner_start)) = '-';
  txt{end+1} = banner_end;

  txt{end+1} = ' ';
  txt{end+1} = ' ';
end

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

downsample

PURPOSE ^

DOWNSAMPLE AOs containing time-series data.

SYNOPSIS ^

function bo = downsample(varargin)

DESCRIPTION ^

```
 DOWNSAMPLE AOs containing time-series data.

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

 DESCRIPTION: DOWNSAMPLE AOs containing time-series data.

 CALL:        b = downsample(a, 4)     - downsample x4; offset is set to default of 0
              b = downsample(a, 2, 1)  - downsample x2 with 1 sample offset
              b = downsample(a, pl)    - use plist to get paramters
              b = downsample(a1, a2, pl) - downsample both a1 and a2; b is then a 2x1
                                     vector.

 PARAMETERS: 'factor'  - decimation factor
             'offset'  - sample offset for decimation [default: 0]

 VERSION:     $Id: downsample.m,v 1.2 2008/02/12 08:31:45 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = downsample(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = downsample(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = downsample(ao,'Category')

 HISTORY: 14-05-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- find FIND particular samples that satisfy the input query and return a new AO.
- len LEN overloads the length operator for Analysis objects. Length of the data samples.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

SUBFUNCTIONS ^

- function pl_default = getDefaultPL()

SOURCE CODE ^

```
function bo = downsample(varargin)
% DOWNSAMPLE AOs containing time-series data.
%
%%%
```

```
% >> category = downsample(ao,'Category')
%
% HISTORY: 14-05-07 M Hewitson
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ALGONAME = mfilename;
CATEGORY = 'Signal Processing';
VERSION  = '$Id: downsample.m,v 1.2 2008/02/12 08:31:45 mauro Exp $';

%% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      bo = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      bo = VERSION;
      return
    elseif strcmp(in, 'Category')
      bo = CATEGORY;
      return
    end
  end
end

invars = {};
as     = [];
ps     = [];
aos_in = 0;

%% collect input ao's, plist's and ao variable names

for j=1:nargin

  if isa(varargin{j}, 'ao')
    as = [as varargin{j}];
    aos_in = aos_in + 1;

    ao_name = inputname(j);
    if isempty(ao_name)
      ao_name = 'no_ao_name';
    end

    % Memorise the variable name of the corresponding analysis object.
    % If the ao is an array or vector add the index to the variable name
    if numel(varargin{j}) == 1
      invars{end+1} = ao_name;
    else
      for ii=1:numel(varargin{j})
```

```
        [I,J] = ind2sub(size(varargin{j}),ii);
        invars{end+1} = sprintf('%s(%d,%d)', ao_name, I, J);
      end
    end
  end
  if isa(varargin{j}, 'plist')
    ps = [ps varargin{j}];
  end
end

Na = numel(as);
if isempty(as)
  error('### Please input at least one AO.');
end

%% Combine plists
if ~isempty(ps)
  pl = combine(ps);
else
  pl = plist();
end

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

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

if nargin-aos_in == 1
  if isnumeric(varargin{end})
    factor = varargin{end};
  end
end
if nargin-aos_in == 2
  if isnumeric(varargin{end-1})
    factor = varargin{end-1};
  end
  if isnumeric(varargin{end})
    offset = varargin{end};
  end
end

if isempty(factor)
  error('### Please specify a decimation factor either directly or in a plist.');
end
if isempty(offset)
  warning('!!! No offset specified; using default of 0 samples !!!');
  offset = 0;
end
```

```

%% Loop over input AOs
bo = [];
for j=1:Na

  a  = as(j);
  d  = a.data;
  x  = d.x;
  y  = d.y;
  fs = d.fs;
  if isa(d, 'tsdata')
    ss = 1+offset;
    samples = ss:factor:len(a);
    d = set(d, 'x', x(samples));
    d = set(d, 'y', y(samples));
    d = set(d, 'fs', fs/factor);
  else
    error('### I can only downsample time-series AOs.');
  end

  %------- Make output AO

  % create new output history
  plo  = plist([param('factor', factor) param('offset', offset)]);
  h = history(ALGONAME, VERSION, plo, a.hist);
  h = set(h, 'invars', invars);

  % make output analysis object
  b = ao(d, h);

  % set name
  b = setnh(b, 'name', sprintf('downsample(%s)', invars{j}));

  % Add to output array
  bo = [bo b];

end

% Reshape the ouput to the same size of the input
bo = reshape(bo, size(as));

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('factor',  '')
                      param('offset',  0)]);


% END
```

eig

PURPOSE ^

EIG overloads the determinant function for Analysis objects.

SYNOPSIS ^

function varargout = eig(varargin)

DESCRIPTION ^

```
 EIG overloads the determinant function for Analysis objects.

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

 DESCRIPTION: EIG overloads the determinant function for Analysis objects.

 CALL:        v     = eig (a,pl)    % only with data = cdata
            [v, d] = eig (a,pl)
            [v, d] = eig (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. 'nobalance' to disable balancing options are the
                               same as for the matlab function.

 VERSION:     $Id: eig.m,v 1.20 2008/02/19 15:23:23 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = eig(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = eig(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = eig(ao,'Category')
```

```
 HISTORY: 08-05-07 A Monsky
             Creation

%%%
```

```
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = eig(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = eig(ao,'Category')
%
% HISTORY: 08-05-07 A Monsky
%             Creation
%
%%%
```

```
%% Loop over analysis objects
for j=1:numel(as)
  a = as(j);

  d = get(a, 'data');
  dinfo = whos('d');
  % Which data type do we have
  dtype = dinfo.class;

  switch dtype
    case 'cdata'
      if size(d.y,1) ~= size(d.y,2)
        error('### The value (y) of data must be a square matrix.')
      else
        %disp('* Eigenvalues of cdata object');
        if nargout <= 1
          [h, vdata] = single_operation(d, 'eig',pl);
        elseif nargout == 2
          [h, vdata, ddata] = single_operation(d, 'eig',pl);
        end
        %create analysis object(s)
        h = set(h, 'inhists', [a.hist]);

        %% Set the var_name to the history
        h = set(h, 'invars', cellstr(invars{j}));

        v = ao(vdata, h);
        v = setnh(v, 'name',  sprintf('v_eig(%s)', char(invars{j})));
        V = [V v];
        if ~isempty(ddata)
          d = ao(ddata, h);
          d = setnh(d, 'name',  sprintf('d_eig(%s)', char(invars{j})));
          D = [D d];
        end
      end
    case {'tsdata','fsdata','xydata'}
      error('### this function works for cdata type AO only')
    otherwise
      error('### unknown data type.')
  end
end

% Reshape the ouput to the same size of the input
V = reshape(V, size(as));
varargout{1} = V;

if nargout > 1
  % Reshape the ouput to the same size of the input
  D = reshape(D, size(as));
  varargout{2} = D;
```

```
0130   if nargout > 2
0131     error('### wrong number of outputs')
0132   end
0133 end
0134
0135 %% Get default params
0136 function plo = getDefaultPL()
0137
0138 plo = plist();
0139 plo = append(plo, 'option', '');
0140
0141 % END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

exp

PURPOSE ^

EXP overloads the exp operator for Analysis objects. Exponential.

SYNOPSIS ^

function ao_out = exp (varargin)

DESCRIPTION ^

```
 EXP overloads the exp operator for Analysis objects. Exponential.

%%%
```

```
              Creation

%%%
```

```
% >> version = exp(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = exp(ao,'Category')
%
% HISTORY: 23-05-2007 Diepholz
%             Creation
%
%%%
```

```
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  %% create a new analysis objects
  new_ao = a;
  new_ao = ao  (exp_data, h);
  new_ao = setnh (new_ao, 'name', sprintf('exp(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('xdata',  '')
                      param('ydata',  '')]);


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

fft

PURPOSE ^

FFT overloads the fft operator for Analysis objects.

SYNOPSIS ^

function varargout = fft(varargin)

DESCRIPTION ^

```
 FFT overloads the fft operator for Analysis objects.

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

 DESCRIPTION: FFT overloads the fft operator for Analysis objects.

 CALL:        b = fft(a)

 VERSION:     $Id: fft.m,v 1.12 2008/02/12 08:31:45 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = fft(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = fft(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = fft(ao,'Category')

 HISTORY: 12-03-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.

- char CHAR overloads char() function for analysis objects.
- fft FFT overloads the fft operator for Analysis objects.
- get GET get ao properties.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

- fft FFT overloads the fft operator for Analysis objects.

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = fft(varargin)
% FFT overloads the fft operator for Analysis objects.
%
%%%
```

```
% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    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
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, ps, invars] = collect_inputs(varargin, in_names);


% Check input analysis object
for j=1:numel(as)
  a     = as(j);
  d     = get(a, 'data');
  dinfo = whos('d');

  % Which data type do we have
  dtype = dinfo.class;
  switch dtype
    case 'tsdata'

      % make a new fsdata object
      nfft = length(d.y);
      ft   = fft(d.y);
      ft   = ft(1:nfft/2+1);
      f    = linspace(0, d.fs/2, length(ft));
      fsd = fsdata(f, ft, d.fs);
      fsd = set(fsd, 'name', sprintf('fft(%s)', d.name));
      fsd = set(fsd, 'xunits', 'Hz');
      fsd = set(fsd, 'yunits', '');

      % make a new history object
      h = history(ALGONAME, VERSION, plist(), [a.hist]);
      h = set(h, 'invars', cellstr(invars{j}));

```

```
      % make output analysis object
      b = ao(fsd, h);
      b = setnh(b, 'name', sprintf('fft(%s)', invars{j}));

    case {'fsdata', 'cdata', 'xydata'}
      error('### I don''t work for frequency-series, xy and constant data yet.');
      disp('* FFT of fsdata object');


    otherwise
      error('### unknown data type. They can not be addded.')
  end

  % 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;


%-----------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

plo = plist();

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

filter

PURPOSE ^

FILTER overides the filter function for analysis objects.

SYNOPSIS ^

function varargout = filter(varargin)

DESCRIPTION ^

```
 FILTER overides the filter function for analysis objects.

%%%
```

```
The following call returns a string that contains the routine CVS version:

>> version = filter(ao,'Version')

The following call returns a string that contains the routine category:

>> category = filter(ao,'Category')

HISTORY: 11-02-07 M Hewitson
            Creation

%%%
```

```
%               then the response of the filter is computed and then multiplied
%               with the input frequency series. The output analysis object
%               contains a frequency series.
%
% CALL:         >> [b, filt] = filter(a,pl)
%               >> [b, filt] = filter(a,filt,pl)
%               >> b = filter(a,pl)
%
% INPUTS:       pl   - a parameter list
%               a    - input analysis object
%
% OUTPUTS:      filt - a copy of the input filter object with the
%                      history values filled in.
%               b    - output analysis object containing the filtered data.
%
% PARAMETERS:   filter - the filter object to use to filter the data
%
% VERSION:      $Id: filter.m,v 1.26 2008/02/12 08:31:45 mauro Exp $
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = filter(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = filter(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = filter(ao,'Category')
%
% HISTORY: 11-02-07 M Hewitson
%             Creation
%
%%%
```

```
      varargout{1} = CATEGORY;
      return
    end
  end
end

%% capture input variable names
fobj   = [];
bs     = [];

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

  if isa(varargin{ii}, 'miir') || isa(varargin{ii}, 'mfir')
    fobj = varargin{ii};
  end
end

[as, ps, invars] = collect_inputs(varargin, in_names);

if isa(ps, 'plist')
  pl = combine(ps);
else
  pl = plist();
end
if isempty(fobj)
  fobj = find(pl, 'filter');
end

% check inputs
if ~isa(fobj, 'miir') && ~isa(fobj, 'mfir')
  error('### the filter input should be an miir/mfir object.');
end

for j=1:numel(as)

  % get input data
  a = as(j);
  d = a.data;

  %--------- Time-series filter
  %
  if isa(d, 'tsdata')

    % get input data
    fs = d.fs;

    if isa(fobj, 'mfir')

      % apply filter
      disp('* filtering with FIR filter...');
```

```
      G  = get(fobj, 'gain');

      %     [fstruct, y] = ltpda_firfilter(struct(filt), d.y);
      coeffs  = get(fobj, 'a');
      Zi      = get(fobj, 'histout');
      [y, Zf] = filter(coeffs, 1, G.*d.y, Zi);

      % remove group delay
      if isempty(find(pl, 'gdoff'))
        gd = floor(get(fobj, 'gd'));
        x = d.x(1:end-gd);
        y = y(1+gd:end);
      else
        x = d.x;
      end

      % consolodate this structure with mfir class before converting.
      fobj = set(fobj, 'histout', Zf.');

    else %if isa(fobj, 'miir')

      if fs ~= get(fobj, 'fs')
        warning('!!! Filter is designed for a different sample rate of data.');
        % Adjust/redesign if this is a standard filter
        fobj = redesign(fobj, fs);
      end

      % apply filter
%       [fstruct, y] = ltpda_iirfilter(struct(fobj), d.y, length(d.y));

      acoeffs  = get(fobj, 'a');
      bcoeffs  = get(fobj, 'b');
      Zi       = get(fobj, 'histout');
      [y, Zf] = filter(acoeffs, bcoeffs, d.y, Zi);
      x = d.x;
      % consolodate this structure with miir class before converting.
%       fobj = set(fobj, 'histin', fstruct.histin);
%       fobj = set(fobj, 'histout', fstruct.histout);
      fobj = set(fobj, 'histout', Zf);
    end

    %----- Create output analysis object
    % make a new tsdata object
    ts = tsdata(y, fs);
    ts = set(ts, 'name', sprintf('filter %s with %s', d.name, get(fobj,'name')));
    ts = set(ts, 'xunits', d.xunits);
    ts = set(ts, 'yunits', d.yunits);
    ts = set(ts, 't0', d.t0);

    % make a new history object
```

```
    pl = plist();
    pl = append(pl, param('filter', fobj));
    h = history(ALGONAME, VERSION, pl, a.hist);
    h = set(h, 'invars', cellstr(invars{j}));

    % make output analysis object
    b = ao(ts, h);
    % name for this object
    b  = setnh(b, 'name', sprintf('%s(%s)', get(fobj,'name'), invars{j}));
    bs = [bs b];

    %--------- Frequency-series filter
    %
  elseif isa(d, 'fsdata')
    error('### I don''t work yet. Please code me up.');

  else
    error('### unknown data type.');
  end
end

% Reshape the ouput to the same size of the input
bs = reshape(bs, size(as));

if nargout == 1
  varargout{1} = bs;
elseif nargout == 2
  varargout{1} = bs;
  varargout{2} = fobj;
else
  error('### wrong number of output arguments.');
end

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist(param('filter',  ''));
  pl_default = append(pl_default, 'gdoff' ,[]);
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

filtfilt

PURPOSE ^

FILTFILT overides the filtfilt function for analysis objects.

SYNOPSIS ^

function varargout = filtfilt(varargin)

DESCRIPTION ^

```
 FILTFILT overides the filtfilt function for analysis objects.

%%%
```

```
>> pl = filtfilt(ao, 'Params')

The following call returns a string that contains the routine CVS version:

>> version = filtfilt(ao,'Version')

The following call returns a string that contains the routine category:

>> category = filtfilt(ao,'Category')

HISTORY: 11-02-07 M Hewitson
            Creation

%%%
```

```
%               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
%
% REMARK:       Uses ltpda_filtfilt() to do the filtering.
%
% VERSION:      $Id: filtfilt.m,v 1.15 2008/02/12 08:31:45 mauro Exp $
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = filtfilt(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = filtfilt(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = filtfilt(ao,'Category')
%
% HISTORY: 11-02-07 M Hewitson
%             Creation
%
%%%
```

```
      varargout{1} = VERSION;
      return
    elseif strcmp(in, 'Category')
      varargout{1} = CATEGORY;
      return
    end
  end
end

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

  if isa(varargin{ii}, 'miir')
    filt = varargin{ii};
  end
end

[as, ps, invars] = collect_inputs(varargin, in_names);

if isa(ps, 'plist')
  pl = combine(ps);
else
  pl = plist();
end

if isempty(filt)
  filt = find(pl, 'filter');
end

% check inputs
if ~isa(filt, 'miir')
  error('### the first input should be an miir object.');
end
if ~isa(as, 'ao')
  error('### second input should be an analysis object.');
end

for ii=1:numel(as)

  % get input data
  a = as(ii);
  d = a.data;

  %--------- Time-series filter
  %
  if isa(d, 'tsdata')

    % get input data
    y  = d.y;
    fs = d.fs;
    if fs ~= get(filt, 'fs')
```

```
      warning('!!! Filter is designed for a different sample rate of data.');
      % Adjust/redesign if this is a standard filter
      filt = redesign(filt, fs);
    end

    % get filter coeffs
    ac = get(filt,'a');
    bc = get(filt,'b');

    % apply filter
    y = filtfilt(ac, bc, y);
    %   [fstruct, y] = ltpda_filtfilt(struct(filt), d.y, length(d.y));
    % consolodate this structure with miir class before converting.
    %   filt = set(filt, 'histin', fstruct.histin);
    %   filt = set(filt, 'histout', fstruct.histout);

    %----- Create output analysis object
    % make a new tsdata object
    ts = tsdata(d.x, y);
    ts = set(ts, 'name', sprintf('filtfilt %s with %s', d.name, get(filt,'name')));
    ts = set(ts, 'xunits', d.xunits);
    ts = set(ts, 'yunits', d.yunits);
    ts = set(ts, 't0', d.t0);

    % make a new history object
    pl = plist();
    pl = append(pl, param('filter', filt));
    h = history(ALGONAME, VERSION, pl, a.hist);
    h = set(h, 'invars', cellstr(invars{ii}));

    % make output analysis object
    bs = ao(ts, h);
    % name for this object
    bs = setnh(bs, 'name', sprintf('%s(%s)', get(filt,'name'), invars{ii}));

    bo = [bo bs];

    %--------- Frequency-series filter
    %
  elseif isa(d, 'fsdata')
    error('### I don''t work yet. Please code me up.');

  else
    error('### unknown data type.');
  end

end

% Reshape the ouput to the same size of the input
bo = reshape(bo, size(as));
```

```

if nargout == 1
  varargout{1} = bo;
elseif nargout == 2
  varargout{1} = bo;
  varargout{2} = filt;
else
  error('### wrong number of output arguments.');
end


%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist(param('filter',  ''));
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

imag

PURPOSE ^

IMAG overloads the imaginary operator for Analysis objects.

SYNOPSIS ^

function ao_out = imag(varargin)

DESCRIPTION ^

```
 IMAG overloads the imaginary operator for Analysis objects.

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

 DESCRIPTION: IMAG overloads the imaginary operator for Analysis objects.

 CALL: ao_out = imag(ao_in);
       ao_out = imag(ao_in, pl);

 POSSIBLE VALUES: ao_in  = [ao2 ao3]
                  ao_in  = ao_vector
                  ao_in  = ao_matrix

 PARAMETER LIST:  <key>           <value>             <description>
                          tsdata  fsdata  xydata
                 'xdata'   't'     'f'      'x'     compute the xdata
                 'ydata'   'x'     'xx'     'y'     compute the ydata

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = imag(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = imag(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = imag(ao,'Category')

 VERSION: $Id: imag.m,v 1.5 2008/02/12 08:31:45 mauro Exp $

 HISTORY: 20-0802007 Diepholz
```

```
              Creation

%%%
```

```
% The following call returns a string that contains the routine CVS version:
%
% >> version = imag(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = imag(ao,'Category')
%
% VERSION: $Id: imag.m,v 1.5 2008/02/12 08:31:45 mauro Exp $
%
% HISTORY: 20-0802007 Diepholz
%             Creation
%
%%%
```

```

  [h, imag_data] = single_operation(a.data, 'imag', pl);

  %% Add the history from the ao object to the history
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  %% create a new analysis objects
  new_ao = ao  (imag_data, h);
  new_ao = setnh(new_ao, 'name', sprintf('imag(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));


%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('xdata',  '')
                      param('ydata',  '')]);


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

interp

PURPOSE ^

INTERP interpolate the values in the input AO(s) at new values specified by the input parameter list.

SYNOPSIS ^

function varargout = interp(varargin)

DESCRIPTION ^

```
 INTERP interpolate the values in the input AO(s) at new values specified by the
input parameter list.

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

 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
                   'cubic'   - shape-preserving piecewise cubic interpolation

 REMARK:      Matrix cdata objects are not supported.

 VERSION:     $Id: interp.m,v 1.11 2008/02/12 08:31:45 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = interp(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = interp(ao,'Version')
```

```
 The following call returns a string that contains the routine category:

 >> category = interp(ao,'Category')

 HISTORY: 05-06-07 M Hewitson
             Creation

%%%
```

```
%                    'cubic'   - shape-preserving piecewise cubic interpolation
%
% REMARK:      Matrix cdata objects are not supported.
%
% VERSION:     $Id: interp.m,v 1.11 2008/02/12 08:31:45 mauro Exp $
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = interp(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = interp(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = interp(ao,'Category')
%
% HISTORY: 05-06-07 M Hewitson
%             Creation
%
%%%
```

```
if isempty(ps)
  pl = getDefaultPL();
else
  pl = combine(ps, getDefaultPL);
end


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

%-----------------------
% Loop over input AOs
bs = [];
for j=1:numel(as)

  a = as(j);

  %----------------------------
  % Get data
  [x,y] = get_xy_values(a.data);

  %----------------------------
  % Interpolate this vector
  if ~isempty(x)
    % for tsdata, fsdata and xydata objects
    y = interp1(x,y,vertices, method, 'extrap');
  else
    % for cdata object
    y = interp1(y,vertices, method, 'extrap');
  end

  %----------------------------
  % Make output AO/cdata
  d = a.data;

  d = set_xy_axis(d, vertices, y.');

  d = set(d, 'name', sprintf('%s(%s)', method, d.name));

  h = history(ALGONAME, VERSION, pl, a.hist);
  h = set(h, 'invars', invars(j));

  % make output analysis object
  b = ao(d, h);

  % set name
  b = setnh(b, 'name', sprintf('%s(%s)', method, invars{j}));

  % add to outputs
  bs = [bs b];
```

```
end

% Reshape the ouput to the same size of the input
bs = reshape(bs, size(as));

% Set outputs
varargout{1} = bs;

%-----------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('vertices', []));
plo = append(plo, param('method', 'spline'));



% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

inv

PURPOSE ^

INV overloads the inverse function for Analysis Objects.

SYNOPSIS ^

function varargout = inv(varargin)

DESCRIPTION ^

```
 INV overloads the inverse function for Analysis Objects.

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

 DESCRIPTION: INV overloads the inverse function for Analysis Objects.

 CALL:        a = inv(a1)   % only with data = cdata

 VERSION:     $Id: inv.m,v 1.16 2008/02/12 19:29:33 hewitson Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = inv(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = inv(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = inv(ao,'Category')

 HISTORY: 08-05-07 A Monsky
            Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.

- char CHAR overloads char() function for analysis objects.
- get GET get ao properties.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = inv(varargin)
% INV overloads the inverse function for Analysis Objects.
%
%%%
```

```
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    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
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, ps, invars] = collect_inputs(varargin, in_names);

% check plist
if isempty(ps)
  pl = getDefaultPL();
else
  pl = combine(ps, getDefaultPL);
end

%% go through analysis objects
for j=1:numel(as)
  a = as(j);
  d = get(a, 'data');
  dinfo = whos('d');

  % Which data type do we have
  dtype = dinfo.class;
  switch dtype
    case 'cdata'
      if size(d.y,1) ~= size(d.y,2)
        error('### The value (y) of data must be a square matrix.')
      else
        %disp('* INVERSE of cdata object');

        % make a new cdata object
        [h, c] = single_operation(d, 'inv', pl);

        % make output analysis object
        h = set(h, 'inhists', [a.hist]);

```

```
        %% Set the var_name to the history
        h = set(h, 'invars', cellstr(invars{j}));

        b = ao(c, h);
        b = setnh(b, 'name',  sprintf('inv(%s)', char(invars{j})));
      end

    case {'tsdata','fsdata','xydata'}
      error('### this function works for cdata type AO only')
    otherwise
      error('### unknown data type.')

  end

  % 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;

%% Get default params
function plo = getDefaultPL()

plo = plist();

% END
```

Generated on Tue 26-Feb-2008 10:52:52 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.

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

 DESCRIPTION: JOIN multiple AOs into a single AO.
              If any two AOs overlap, then the values from the first appear
              in the output AO.

 CALL:        b = join(a1,a2,pl)

 VERSION:     $Id: join.m,v 1.14 2008/02/24 13:43:40 hewitson Exp $

 REMARK:      Input AOs should be of the same type; if not, only AOs of the
              type of the first input AO will be joined together to produce
              the output.

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = join(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = join(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = join(ao,'Category')

 HISTORY: 03-06-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- find FIND particular samples that satisfy the input query and return a new AO.
- set SET set an analysis object property.

This function is called by:

SUBFUNCTIONS ^

- function Toff = getToff(as)
- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = join(varargin)
% JOIN multiple AOs into a single AO.
%
%%%
```

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

```
yo     = [];
fs     = -1;
dname  = '';
aname  = '';
xunits = '';
yunits = '';

% Compute time offset for tsdata objects
% to avoid rounding errors later

Toff = getToff(as);

% loop over AOs
for j=1:numel(as)

  a = as(j);

  % Only get the data type we want
  if isa(a.data, dtype)

    switch dtype
      case 'tsdata'

        % here we concatonate time-series
        [x,y] = get_xy_values(a.data);
        t0 = (a.data.t0.utc_epoch_milli - Toff)/1000;

        % make proper time vector
        x = x + t0;

        % only add samples past the end of existing
        if isempty(xo)
          yo = y;
          xo = x;
        else
          idxPost = find(x > max(xo));
          idxPre  = find(x < min(xo));
          xo = [x(idxPre);xo;x(idxPost)];
          yo = [y(idxPre);yo;y(idxPost)];
        end

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

        % store xunits
        if ~strcmp(xunits, a.data.xunits) && ~isempty(xunits)
          warning('!!! Data has different X units !!!');
        end
        xunits = a.data.xunits;
```

```

        % store yunits
        if ~strcmp(yunits, a.data.yunits) && ~isempty(yunits)
          warning('!!! Data has different Y units !!!');
        end
        yunits = a.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 a.hist];

    % Collect names, invars
    dname = [dname ',' a.data.name];
    if ~isempty(aname)
      if ~strcmp(aname, a.name)
        aname = [aname ',' a.name];
      end
    else
      aname = a.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);
    data = set(data, 't0', (Toff+t0));
```

```
    data = set(data, 'name', dname);
    data = set(data, 'xunits', xunits);
    data = set(data, 'yunits', yunits);

  case 'fsdata'

  case 'xydata'

  case 'cdata'

end

%-----------------------------------------------
% Build output AO

h = history(ALGONAME, VERSION, pl, histin);
h = set(h, 'invars', invars);
a = ao(data, h);
a.name = aname;
%-----------------------------------------------
% set output

varargout{1} = a;


%-------------------------------------------------------------------------
% 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

%-------------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

plo = plist();



% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

log

PURPOSE ^

LOG overloads the log operator for Analysis objects. Natural logarithm.

SYNOPSIS ^

function ao_out = log (varargin)

DESCRIPTION ^

```
 LOG overloads the log operator for Analysis objects. Natural logarithm.

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

 DESCRIPTION: LOG overloads the log operator for Analysis objects.
              Natural logarithm.
              LOG(ao) is the natural logarithm of the elements of ao.data.

 CALL: ao_out = log(ao_in);
       ao_out = log(ao_in, pl);
       ao_out = log(ao1, pl1, ao_vector, ao_matrix, pl2);

 POSSIBLE VALUES: ao_in  = [ao2 ao3]
                  ao_in  = ao_vector
                  ao_in  = ao_matrix

 PARAMETER LIST:  <key>           <value>             <description>
                          tsdata  fsdata  xydata
                 'xdata'   't'     'f'      'x'     compute the xdata
                 'ydata'   'x'     'xx'     'y'     compute the ydata

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = log(ao,'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = log(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = log(ao,'Category')
```

```
 HISTORY: 23-05-2007 Diepholz
             Creation

%%%
```

```
% >> pl = log(ao,'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = log(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = log(ao,'Category')
%
% HISTORY: 23-05-2007 Diepholz
%             Creation
%
%%%
```

```

  [h, log_data] = single_operation(a.data, 'log', pl);

  %% Add the history from the ao object to the history
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  %% create a new analysis objects
  new_ao = ao  (log_data, h);
  new_ao = setnh(new_ao, 'name', sprintf('log(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('xdata',  '')
                      param('ydata',  '')]);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

mean

PURPOSE ^

MEAN overloads the mean operator for Analysis objects. Compute the mean value.

SYNOPSIS ^

function ao_out = mean (varargin)

DESCRIPTION ^

```
 MEAN overloads the mean operator for Analysis objects. Compute the mean value.

%%%
```

```
 >> category = mean(ao,'Category')

 HISTORY: 23-05-2007 Diepholz
             Creation

%%%
```

```
axis
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = mean(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = mean(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = mean(ao,'Category')
%
% HISTORY: 23-05-2007 Diepholz
%             Creation
%
%%%
```

```

[ao_set, pl, invars] = collect_inputs(varargin, in_names);

pl = combine(pl, ps);

%% go through analysis objects
for j=1:numel(ao_set)
  a = ao_set(j);

  [h, mean_data] = single_operation(a.data, 'mean', pl);

  %% Add the history from the ao object to the history
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  % convert to cdata type
  mean_x    = [];
  mean_y    = [];

  do_xdata = find(pl, 'xdata');
  do_ydata = find(pl, 'ydata');

  %% Is no axis entry in the parameter list
  %% then set the default axis = xdata
  if isempty(do_xdata) && isempty(do_ydata)
    do_ydata = 'yes';
  end

  %% Get the operation result. It is stored in the individual axis
  pl_get_axis = plist([param('xdata', do_xdata) param('ydata', do_ydata)]);
  [mean_x, mean_y] = get_xy_values(mean_data, pl_get_axis);

  if ~isempty(do_xdata) && ~isempty(do_ydata)
    mean_cdata = cdata([mean_x mean_y]);
  elseif ~isempty(do_xdata)
    mean_cdata = cdata(mean_x);
  else
    mean_cdata = cdata(mean_y);
  end

  mean_cdata = set (mean_cdata, 'name', mean_data.name);
  mean_cdata = set (mean_cdata, 'xunits', mean_data.xunits);
  mean_cdata = set (mean_cdata, 'yunits', mean_data.yunits);

  %% create a new analysis objects
  new_ao = ao  (mean_cdata, h);
  new_ao = setnh(new_ao, 'name', sprintf('mean(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

```

```
end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));


%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('dim',    '')
                      param('xdata',  '')
                      param('ydata',  '')]);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

median

PURPOSE ^

MEDIAN overloads the median operator for Analysis objects. Compute the median value.

SYNOPSIS ^

function ao_out = median (varargin)

DESCRIPTION ^

```
 MEDIAN overloads the median operator for Analysis objects. Compute the median value.

%%%
```

```
 >> category = median(ao,'Category')

 HISTORY: 23-05-2007 Diepholz
             Creation

%%%
```

```
%                 'ydata'   'x'     'xx'     'y'     compute the median of the y-axis
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = median(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = median(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = median(ao,'Category')
%
% HISTORY: 23-05-2007 Diepholz
%             Creation
%
%%%
```

```
end

[ao_set, pl, invars] = collect_inputs(varargin, in_names);

pl = combine(pl, ps);

%% go through analysis objects
for j=1:numel(ao_set)
  a = ao_set(j);

  [h, median_data] = single_operation(a.data, 'median', pl);

  %% Add the history from the ao object to the history
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  do_xdata = find(pl, 'xdata');
  do_ydata = find(pl, 'ydata');

  %% Is no axis entry in the parameter list
  %% then set the default axis = xdata
  if isempty(do_xdata) && isempty(do_ydata)
    do_ydata = 'yes';
  end

  %% Get the operation result. It is stored in the individual axis
  pl_get_axis = plist([param('xdata', do_xdata) param('ydata', do_ydata)]);
  [median_x, median_y] = get_xy_values(median_data, pl_get_axis);

  if ~isempty(do_xdata) && ~isempty(do_ydata)
    median_cdata = cdata([median_x median_y]);
  elseif ~isempty(do_xdata)
    median_cdata = cdata(median_x);
  else
    median_cdata = cdata(median_y);
  end

  median_cdata = set (median_cdata, 'name', median_data.name);
  median_cdata = set (median_cdata, 'xunits', median_data.xunits);
  median_cdata = set (median_cdata, 'yunits', median_data.yunits);

  %% create a new analysis objects
  new_ao = a;
  new_ao = ao  (median_cdata, h);
  new_ao = setnh(new_ao, 'name', sprintf('median(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

end

```

```
% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));


%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('dim',    '')
                      param('xdata',  '')
                      param('ydata',  '')]);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

minus

PURPOSE ^

MINUS overloads - operator for analysis objects.

SYNOPSIS ^

function b = minus(varargin)

DESCRIPTION ^

```
 MINUS overloads - operator for analysis objects.

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

 DESCRIPTION: MINUS overloads - operator for analysis objects.

 CALL:        a = a1-a2

 VERSION:     $Id: minus.m,v 1.12 2008/02/13 18:35:22 hewitson Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = minus(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = minus(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = minus(ao,'Category')

 HISTORY: 05-02-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.

- char CHAR overloads char() function for analysis objects.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

SUBFUNCTIONS ^

- function pl_default = getDefaultPL()

SOURCE CODE ^

```
function b = minus(varargin)
% MINUS overloads - operator for analysis objects.
%
%%%
```

```
      b = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      b = VERSION;
      return
    elseif strcmp(in, 'Category')
      b = CATEGORY;
      return
    end
  end
end


%% capture input variable names
invars = {};
for j=1:nargin
  iname = inputname(j);
  if isempty(iname) && isnumeric(varargin{j})
    iname = num2str(varargin{j});
  elseif isempty(iname) && ischar(varargin{j})
    iname = varargin{j};
  end
  invars = [invars cellstr(iname)];
end

op = '-';

[a1,a2,do] = aooperate(varargin, op);

%-------- create output AO

% make a new history object
h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
h = set(h, 'invars', invars);

% get names for output
if isempty(char(invars{1}))
  n1 = a1.name;
else
  n1 = char(invars{1});
end
if isempty(char(invars{2}))
  n2 = a2.name;
else
  n2 = char(invars{2});
end


% make output analysis object
b = ao(do, h);
b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));
```

```
0090
0091
0092
0093 %% Get default params
0094 function pl_default = getDefaultPL()
0095
0096   pl_default = plist();
0097
0098 % END
0099
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

mpower

PURPOSE ^

MPOWER implements power operator for analysis objects.

SYNOPSIS ^

function b = mpower(varargin)

DESCRIPTION ^

```
 MPOWER implements power operator for analysis objects.

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

 DESCRIPTION: MPOWER implements power operator for analysis objects.
              computes b^c where b is an analysis object and c is either
              another analysis object or a constant.

 CALL:        a = a1^2
              a = a1^a2     % with a2.data.vals = scalar

 VERSION:     $Id: mpower.m,v 1.13 2008/02/13 18:35:22 hewitson Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = mpower(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = mpower(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = mpower(ao,'Category')

 HISTORY: 01-02-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

SUBFUNCTIONS ^

- function pl_default = getDefaultPL()

SOURCE CODE ^

```
function b = mpower(varargin)
% MPOWER implements power operator for analysis objects.
%
%%%
```

```

%% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      b = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      b = VERSION;
      return
    elseif strcmp(in, 'Category')
      b = CATEGORY;
      return
    end
  end
end

%% capture input variable names
invars = {};
for j=1:nargin
  iname = inputname(j);
  if isempty(iname) & isnumeric(varargin{j})
    iname = num2str(varargin{j});
  elseif isempty(iname) & ischar(varargin{j})
    iname = varargin{j};
  end
  invars = [invars cellstr(iname)];
end

op = '^';

[a1,a2,do] = aooperate(varargin, op);

%--------- create output AO

% make a new history object
h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
h = set(h, 'invars', invars);

% get names for output
if isempty(char(invars{1}))
  n1 = a1.name;
else
  n1 = char(invars{1});
end
if isempty(char(invars{2}))
  n2 = a2.name;
else
  n2 = char(invars{2});
end
```

```

% make output analysis object
% op = op(2:end);
b = ao(do, h);
b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));


%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist();


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

mrdivide

PURPOSE ^

MRDIVIDE overloads / operator for analysis objects.

SYNOPSIS ^

function b = mrdivide(varargin)

DESCRIPTION ^

```
 MRDIVIDE overloads / operator for analysis objects.

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

 DESCRIPTION: MRDIVIDE overloads / operator for analysis objects.

 CALL:        a = a1/scalar
              a = a1/a2

 VERSION:     $Id: mrdivide.m,v 1.13 2008/02/13 18:35:22 hewitson Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = mrdivide(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = mrdivide(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = mrdivide(ao,'Category')

 HISTORY: 05-02-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

SUBFUNCTIONS ^

- function pl_default = getDefaultPL()

SOURCE CODE ^

```
function b = mrdivide(varargin)
% MRDIVIDE overloads / operator for analysis objects.
%
%%%
```

```
    in = char(varargin{2});
    if strcmp(in, 'Params')
      b = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      b = VERSION;
      return
    elseif strcmp(in, 'Category')
      b = CATEGORY;
      return
    end
  end
end

%% capture input variable names
invars = {};
for j=1:nargin
  iname = inputname(j);
  if isempty(iname) && isnumeric(varargin{j})
    iname = num2str(varargin{j});
  elseif isempty(iname) && ischar(varargin{j})
    iname = varargin{j};
  end
  invars = [invars cellstr(iname)];
end

op = '/';

[a1,a2,do] = aooperate(varargin, op);

%-------- create output AO

% make a new history object
h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
h = set(h, 'invars', invars);

% get names for output
if isempty(char(invars{1}))
  n1 = a1.name;
else
  n1 = char(invars{1});
end
if isempty(char(invars{2}))
  n2 = a2.name;
else
  n2 = char(invars{2});
end

% make output analysis object
% op = op(2:end);
b = ao(do, h);
```

```
b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist();


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

mtimes

PURPOSE ^

MTIMES overloads * operator for analysis objects.

SYNOPSIS ^

function b = mtimes(varargin)

DESCRIPTION ^

```
 MTIMES overloads * operator for analysis objects.

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

 DESCRIPTION: MTIMES overloads * operator for analysis objects.
              This just does the same as times (*) for analysis objects.

 CALL:        a = a1*scalar
              a = a1*a2

 VERSION:     $Id: mtimes.m,v 1.14 2008/02/13 18:35:22 hewitson Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = mtimes(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = mtimes(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = mtimes(ao,'Category')

 HISTORY: 05-02-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

SUBFUNCTIONS ^

- function pl_default = getDefaultPL()

SOURCE CODE ^

```
function b = mtimes(varargin)
% MTIMES overloads * operator for analysis objects.
%
%%%
```

```
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      b = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      b = VERSION;
      return
    elseif strcmp(in, 'Category')
      b = CATEGORY;
      return
    end
  end
end

% capture input variable names
invars = {};
for j=1:nargin
  iname = inputname(j);
  if isempty(iname) && isnumeric(varargin{j})
    iname = num2str(varargin{j});
  elseif isempty(iname) && ischar(varargin{j})
    iname = varargin{j};
  end
  invars = [invars cellstr(iname)];
end

op = '*';

[a1,a2,do] = aooperate(varargin, op);

%--------- create output AO

% make a new history object
h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
h = set(h, 'invars', invars);

% get names for output
if isempty(char(invars{1}))
  n1 = a1.name;
else
  n1 = char(invars{1});
end
if isempty(char(invars{2}))
  n2 = a2.name;
else
  n2 = char(invars{2});
end

% make output analysis object
% op = op(2:end);
```

```
b = ao(do, h);
b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist();


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

norm

PURPOSE ^

NORM overloads the norm operator for Analysis Objects.

SYNOPSIS ^

function varargout = norm(varargin)

DESCRIPTION ^

```
 NORM overloads the norm operator for Analysis Objects.

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

 DESCRIPTION:NORM overloads the norm operator for Analysis Objects.

 CALL:        b = norm (a,pl) % only with data = cdata
              b = norm (a)

 INPUTS:      pl   - a parameter list
              a    - input analysis object

 OUTPUTS:     like matlab fct

 Parameters: 'option' - a string or value that can be 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

 VERSION:     $Id: norm.m,v 1.16 2008/02/12 09:29:24 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = norm(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = norm(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = norm(ao,'Category')

 HISTORY: 08-05-07 A Monsky
```

```
              Creation

%%%
```

```
%
% >> version = norm(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = norm(ao,'Category')
%
% HISTORY: 08-05-07 A Monsky
%             Creation
%
%%%
```

```
  d = get(a, 'data');
  dinfo = whos('d');
  % Which data type do we have
  dtype = dinfo.class;
  switch dtype
    case 'cdata'
      %disp('* NORM of cdata object');
      % make a new cdata object
      [h, c] = single_operation(d, 'norm',pl);

      % make output analysis object
      h = set(h, 'inhists', [a.hist]);

      %% Set the var_name to the history
      h = set(h, 'invars', cellstr(invars{j}));

      b = ao(c, h);
      b = setnh(b, 'name',  sprintf('norm(%s)', char(invars{j})));
    case {'tsdata', 'fsdata', 'xydata'}
      error('### this function works for cdata type AO only')
    otherwise
      error('### unknown data type.')

  end

  % 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;

%% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, 'option', []);


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

phase

PURPOSE ^

PHASE overloads the ltpda_phase operator for Analysis objects.

SYNOPSIS ^

function ao_out = phase(varargin)

DESCRIPTION ^

```
 PHASE overloads the ltpda_phase operator for Analysis objects.

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

 DESCRIPTION: PHASE overloads the ltpda_phase operator for Analysis objects.

 CALL:            ao_out = phase(ao_in);
                  ao_out = phase(ao_in, pl);

 POSSIBLE VALUES: ao_in  = [ao2 ao3]
                  ao_in  = ao_vector
                  ao_in  = ao_matrix

 PARAMETER LIST:  <key>           <value>             <description>
                          tsdata  fsdata  xydata
                 'xdata'   't'     'f'      'x'     compute the xdata
                 'ydata'   'x'     'xx'     'y'     compute the ydata

 VERSION:         $Id: phase.m,v 1.6 2008/02/12 09:29:24 mauro Exp $

 HISTORY:         22-08-2007 M Hewitson
                     Creation

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = phase(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = phase(ao,'Version')

 The following call returns a string that contains the routine category:
```

```
>> category = phase(ao,'Category')

%%%
```

```
%
% >> pl = phase(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = phase(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = phase(ao,'Category')
%
%%%
```

```
  phase_data = set(phase_data, 'name',   sprintf ('phase(%s)',a.data.name));

  %% Add the history from the ao object to the history
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  %% create a new analysis objects
  new_ao = ao  (phase_data, h);
  new_ao = setnh(new_ao, 'name', sprintf('phase(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));


%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('xdata',  '')
                      param('ydata',  '')]);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

plus

PURPOSE ^

PLUS implements addition operator for two analysis objects.

SYNOPSIS ^

function b = plus(varargin)

DESCRIPTION ^

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

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

 DESCRIPTION: PLUS implements addition operator for two analysis objects.

 CALL:        a = a1+scalar
              a = a1+a2

 VERSION:     $Id: plus.m,v 1.16 2008/02/13 18:35:22 hewitson Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = plus(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = plus(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = plus(ao,'Category')

 HISTORY: 01-02-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

SUBFUNCTIONS ^

- function pl_default = getDefaultPL()

SOURCE CODE ^

```
function b = plus(varargin)
% PLUS implements addition operator for two analysis objects.
%
%%%
```

```
    if strcmp(in, 'Params')
      b = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      b = VERSION;
      return
    elseif strcmp(in, 'Category')
      b = CATEGORY;
      return
    end
  end
end

%% capture input variable names
invars = {};
for j=1:nargin
  iname = inputname(j);
  if isempty(iname) && isnumeric(varargin{j})
    iname = num2str(varargin{j});
  elseif isempty(iname) && ischar(varargin{j})
    iname = varargin{j};
  end
  invars = [invars cellstr(iname)];
end

ALGONAME = mfilename;
VERSION  = '$Id: plus.m,v 1.16 2008/02/13 18:35:22 hewitson Exp $';

op = '+';

[a1,a2,do] = aooperate(varargin, op);

%--------- create output AO

% make a new history object
h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
h = set(h, 'invars', invars);

% get names for output
if isempty(char(invars{1}))
  n1 = a1.name;
else
  n1 = char(invars{1});
end
if isempty(char(invars{2}))
  n2 = a2.name;
else
  n2 = char(invars{2});
end
```

```
% make output analysis object
b = ao(do, h);
b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));


%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist();

% END
```

Generated on Tue 26-Feb-2008 10:52:52 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.

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

 DESCRIPTION: POLYFIT overloads polyfit() function of MATLAB for Analysis
              Objects.

 CALL:        b = polyfit(a, pl)

 VERSION:     $Id: polyfit.m,v 1.11 2008/02/12 09:43:32 mauro Exp $

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

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = polyfit(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = polyfit(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = polyfit(ao,'Category')

 HISTORY: 05-06-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- find FIND particular samples that satisfy the input query and return a new AO.
- polyfit POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

- polyfit POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = polyfit(varargin)
% POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
%
%%%
```

```
% HISTORY: 05-06-07 M Hewitson
%             Creation
%
%%%
```

```

  a = as(j);

  %----------------------------
  % Get data
  [x,y] = get_xy_values(a.data);

  %----------------------------
  % Fit polynomial

  if isempty(coeffs)
    %[p,s,mu] = polyfit(x,y,N);
    [p,s] = polyfit(x,y,N);

    % Add coefficients to plist
    pl = pset(pl, 'coeffs', p);
  else
    p = coeffs;
  end

  % make new values from these coefficients
  y = polyval(p, x);

  %----------------------------
  % Make output AO/cdata
  d = a.data;
  d = set_xy_axis(d, x, y);
  d = set(d, 'name', sprintf('polyfit(%s)', d.name));

  h = history(ALGONAME, VERSION, pl, a.hist);
  h = set(h, 'invars', invars(j));

  % make output analysis object
  b = ao(d, h);

  % set name
  b = setnh(b, 'name', sprintf('polyfit(%s)', invars{j}));

  % add to outputs
  bs = [bs b];

end

% Reshape the ouput to the same size of the input
bs = reshape(bs, size(as));

%---------------------
% Set outputs
varargout{1} = bs;

%-----------------------------------------------------------------------
% Get default params
```

```
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('N', 1));
plo = append(plo, param('coeffs', []));

% END

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

power

PURPOSE ^

POWER implements power operator for analysis objects.

SYNOPSIS ^

function b = power(varargin)

DESCRIPTION ^

```
 POWER implements power operator for analysis objects.

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

 DESCRIPTION: POWER implements power operator for analysis objects.
              computes b.^c where b is an analysis object and c is
              either another analysis object or a constant.

 CALL:        a = a1.^scalar

 VERSION:     $Id: power.m,v 1.11 2008/02/15 16:27:49 hewitson Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = power(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = power(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = power(ao,'Category')

 HISTORY: 01-02-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

SUBFUNCTIONS ^

- function pl_default = getDefaultPL()

SOURCE CODE ^

```
function b = power(varargin)
% POWER implements power operator for analysis objects.
%
%%%
```

```
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      b = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      b = VERSION;
      return
    elseif strcmp(in, 'Category')
      b = CATEGORY;
      return
    end
  end
end

%% capture input variable names
invars = {};
for j=1:nargin
  iname = inputname(j);
  if isempty(iname) & isnumeric(varargin{j})
    iname = num2str(varargin{j});
  elseif isempty(iname) & ischar(varargin{j})
    iname = varargin{j};
  end
  invars = [invars cellstr(iname)];
end

op = '.^';

[a1,a2,do] = aooperate(varargin, op);

%--------- create output AO

% make a new history object
h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
h = set(h, 'invars', invars);

% get names for output
if isempty(char(invars{1}))
  n1 = a1.name;
else
  n1 = char(invars{1});
end
if isempty(char(invars{2}))
  n2 = a2.name;
else
  n2 = char(invars{2});
end

% make output analysis object
```

```
% op = op(2:end);
b = ao(do, h);
b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist();


% END

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

rdivide

PURPOSE ^

RDIVIDE overloads ./ operator for analysis objects.

SYNOPSIS ^

function b = rdivide(varargin)

DESCRIPTION ^

```
 RDIVIDE overloads ./ operator for analysis objects.

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

 DESCRIPTION: RDIVIDE overloads ./ operator for analysis objects.

 CALL:        a = a1./scalar

 VERSION:     $Id: rdivide.m,v 1.13 2008/02/13 18:35:22 hewitson Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = rdivide(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = rdivide(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = rdivide(ao,'Category')

 HISTORY: 05-02-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.

- char CHAR overloads char() function for analysis objects.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

SUBFUNCTIONS ^

- function pl_default = getDefaultPL()

SOURCE CODE ^

```
function b = rdivide(varargin)
% RDIVIDE overloads ./ operator for analysis objects.
%
%%%
```

```
      b = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      b = VERSION;
      return
        elseif strcmp(in, 'Category')
    b = CATEGORY;
    return
    end
  end
end

%% capture input variable names
invars = {};
for j=1:nargin
  iname = inputname(j);
  if isempty(iname) && isnumeric(varargin{j})
    iname = num2str(varargin{j});
  elseif isempty(iname) && ischar(varargin{j})
    iname = varargin{j};
  end
  invars = [invars cellstr(iname)];
end

op = './';

[a1,a2,do] = aooperate(varargin, op);

%--------- create output AO

% make a new history object
h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
h = set(h, 'invars', invars);

% get names for output
if isempty(char(invars{1}))
  n1 = a1.name;
else
  n1 = char(invars{1});
end
if isempty(char(invars{2}))
  n2 = a2.name;
else
  n2 = char(invars{2});
end

% make output analysis object
% op = op(2:end);
b = ao(do, h);
b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));

```

```
%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist();

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

real

PURPOSE ^

REAL overloads the real operator for Analysis objects.

SYNOPSIS ^

function ao_out = real(varargin)

DESCRIPTION ^

```
 REAL overloads the real operator for Analysis objects.

%%%
```

```
                Creation

%%%
```

```
% The following call returns a string that contains the routine CVS version:
%
% >> version = real(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = real(ao,'Category')
%
% VERSION: $Id: real.m,v 1.5 2008/02/12 09:43:32 mauro Exp $
%
% HISTORY: 20-0802007 Diepholz
%             Creation
%
%%%
```

```

  [h, real_data] = single_operation(a.data, 'real', pl);

  %% Add the history from the ao object to the history
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  %% create a new analysis objects
  new_ao = ao  (real_data, h);
  new_ao = setnh(new_ao, 'name', sprintf('real(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));


%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('xdata',  '')
                      param('ydata',  '')]);


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

resample

PURPOSE ^

RESAMPLE overloads resample function for AOs.

SYNOPSIS ^

function bs = resample(varargin)

DESCRIPTION ^

```
 RESAMPLE overloads resample function for AOs.

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

 DESCRIPTION: RESAMPLE overloads resample function for AOs.

 CALL:        b = resample(a, pl)

 Parameters: 'fsout' - specify the desired output sample rate

 VERSION:     $Id: resample.m,v 1.14 2008/02/12 09:43:32 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = resample(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = resample(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = resample(ao,'Category')

 HISTORY: 19-03-07 M Hewitson
            Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- find FIND particular samples that satisfy the input query and return a new AO.
- resample RESAMPLE overloads resample function for AOs.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

- resample RESAMPLE overloads resample function for AOs.

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function bs = resample(varargin)
% RESAMPLE overloads resample function for AOs.
%
%%%
```

```
CATEGORY = 'Signal Processing';

bs      = [];

%% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      bs = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      bs = VERSION;
      return
    elseif strcmp(in, 'Category')
      bs = CATEGORY;
      return
    end
  end
end

ALGONAME = mfilename;
VERSION  = '$Id: resample.m,v 1.14 2008/02/12 09:43:32 mauro Exp $';

%% capture input variables
invars = {};
as     = [];
pl     = [];

for j=1:nargin
  a = varargin{j};
  if isa(a, 'ao')
    d = a.data;
    if isa(d, 'tsdata')
      as = [as a];

      ao_name = inputname(j);
      if isempty(ao_name)
        ao_name = 'no_ao_name';
      end

      % Memorise the variable name of the corresponding analysis object.
      % If the ao is an array or vector add the index to the variable name
      if numel(varargin{j}) == 1
        invars{end+1} = ao_name;
      else
        for ii=1:numel(varargin{j})
          [I,J] = ind2sub(size(varargin{j}),ii);
          invars{end+1} = sprintf('%s(%d,%d)', ao_name, I, J);
        end
      end
    else
```

```
      warning('!!! ignoring non-time-series AO.');
    end
  end
  if isa(a, 'plist')
    pl = a;
  end
end


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');

for j=1:numel(as)

  d  = as(j).data;

  % resample y
  if isempty(filt)
    [y,b] = resample(d.y, fsout, d.fs);
  else
    [p,q] = rat( fsout/d.fs, 1e-12 );
    b = p*filt.a;
    y = resample(d.y, fsout, d.fs, b);
  end


  % make new output tsdata
  do = tsdata(y, fsout);
  d = set(d, 'x', do.x);
  d = set(d, 'y', do.y);
  d = set(d, 'fs', do.fs);

  %--------- create output AO

  % make a new history object
  h = history(ALGONAME, VERSION, pl, as(j).hist);
  h = set(h, 'invars', invars);

  % make output analysis object
  b = ao(d, h);
  b = setnh(b, 'name', sprintf('resample(%s)', invars{j}));
```

```
0136
0137   bs = [bs b];
0138
0139 end
0140
0141 % Reshape the ouput to the same size of the input
0142 bs = reshape(bs, size(as));
0143
0144 % Get default params
0145 function plo = getDefaultPL()
0146
0147 plo = plist();
0148 plo = append(plo, param('fsout', 100));
0149
0150 % END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

select

PURPOSE ^

SELECT select particular samples from the input AOs and return new AOs with only those samples.

SYNOPSIS ^

function bo = 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

 VERSION:     $Id: select.m,v 1.16 2008/02/12 09:43:32 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = select(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = select(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = select(ao,'Category')

 HISTORY: 14-05-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- find FIND particular samples that satisfy the input query and return a new AO.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function bo = select(varargin)
% SELECT select particular samples from the input AOs and return new AOs with only those samples.
%
%%%
```

```
%%%
```

```
samples = [samples find(pl, 'samples')];

% Loop over input AOs
bo = [];
for j=1:numel(as)

  a = as(j);
  d = a.data;

  [x,y] = get_xy_values(d);

  if isa(d, 'cdata')
    if length(d.x) == length(d.y)
      d = set_xy_axis(d, x(samples), y(samples));
    else
      d = set_xy_axis(d, x, y(samples));
    end
  else
    d = set_xy_axis(d, x(samples), y(samples));
  end

  % Make output AO

  % create new output history
  h = history(ALGONAME, VERSION, plist(param('samples', samples)), a.hist);
  h = set(h, 'invars', invars(j));

  % make output analysis object
  b = ao(d, h);

  % set name
  % name for this object
  b = setnh(b, 'name', sprintf('select(%s)', invars{j}));

  % Add to output array
  bo = [bo b];

end

% Reshape the ouput to the same size of the input
bo = reshape(bo, size(as));

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('samples', []));

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

sin

PURPOSE ^

SIN overloads the sin operator for Analysis objects. Sine of argument in radians.

SYNOPSIS ^

function ao_out = sin (varargin)

DESCRIPTION ^

```
 SIN overloads the sin operator for Analysis objects. Sine of argument in radians.

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

 DESCRIPTION: SIN overloads the sin operator for Analysis objects.
              Sine of argument in radians.
              SIN(ao) is the sine of the elements of ao.data.

 CALL: ao_out = sin(ao_in);
       ao_out = sin(ao_in, pl);
       ao_out = sin(ao1, pl1, ao_vector, ao_matrix, pl2);

 POSSIBLE VALUES: ao_in  = [ao2 ao3]
                  ao_in  = ao_vector
                  ao_in  = ao_matrix

 PARAMETER LIST:  <key>           <value>             <description>
                          tsdata  fsdata  xydata
                 'xdata'   't'     'f'      'x'     compute the xdata
                 'ydata'   'x'     'xx'     'y'     compute the ydata

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = sin(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = sin(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = sin(ao,'Category')
```

```
 VERSION: $Id: sin.m,v 1.18 2008/02/12 10:11:13 mauro Exp $

 HISTORY: 07-05-2007 Diepholz
             Creation

 SEE ALSO: ao/cos, ao/tan, ao/asin, acos, atan, Contents

 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>

%%%
```

```
%                 'xdata'   't'     'f'      'x'     compute the xdata
%                 'ydata'   'x'     'xx'     'y'     compute the ydata
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = sin(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = sin(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = sin(ao,'Category')
%
% VERSION: $Id: sin.m,v 1.18 2008/02/12 10:11:13 mauro Exp $
%
% HISTORY: 07-05-2007 Diepholz
%             Creation
%
% SEE ALSO: ao/cos, ao/tan, ao/asin, acos, atan, Contents
%
% 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>
%
%%%
```

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

[ao_set, pl, invars] = collect_inputs(varargin, in_names);

if ~isempty (pl)
  pl = combine(pl);
end

%% go through analysis objects
for j=1:numel(ao_set)
  a = ao_set(j);

  [h, sin_data] = single_operation(a.data, 'sin', pl);

  %% Add the history from the ao object to the history
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  %% create a new analysis objects
  new_ao = ao  (sin_data, h);
  new_ao = setnh(new_ao, 'name', sprintf('sin(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));


%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('xdata',  '')
                      param('ydata',  '')]);


% END
```

Generated on Tue 26-Feb-2008 10:52:52 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.

%%%
```

```
%%%
```

```
% >> version = spectrogram(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = spectrogram(ao,'Category')
%
% HISTORY: 12-03-07 M Hewitson
%             Creation
%
%%%
```

```
    nfft = a.data.fs;
  end
  win  = find(pl, 'Win');
  if length(win.win) < nfft
    switch win.name
      case {'Kaiser', 'Flattop'}
        win = specwin(win.name, nfft, win.psll);
      otherwise
        win = specwin(win.name, nfft);
    end
    disp(sprintf('! Reset window to %s(%d)', strrep(win.name, '_', '\_'), 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 a new history object
  h = history(ALGONAME, VERSION, pl, [a.hist]);
  h = set(h, 'invars', cellstr(invars{j}));

  % Make output AO
  do = xyzdata(T, F, P);
  do = set(do, 'xunits', 's');
  do = set(do, 'yunits', 'Hz');
  do = set(do, 'zunits', [a.data.yunits '^2/Hz']);

  b = ao(do);
  b = setnh(b, 'name', sprintf('specgram(%s)', invars{j}));

  % 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;


%-----------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

w = specwin('Kaiser', 10, 200);
plo = plist('Win', w, 'Nolap', -1, 'Nfft', -1);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

split

PURPOSE ^

SPLIT split an analysis object into the specified segments.

SYNOPSIS ^

function bo = split(varargin)

DESCRIPTION ^

```
 SPLIT split an analysis object into the specified segments.

%%%
```

```
              'interval'    'timespan'
                            - the start/end time are specified in the time span
                              object.

              The UTC time format is: 'yyyy-mm-dd HH:MM:SS'

              If more than one splitting method is specified, the priority
              goes like the list above.

              The time vector in the output AO retains the original
              time values (i.e. it doesn't start from zero).

              The splitting is done as  s<=t<e.

              Arrays of start/stop values should be like: [s1 e1 s2 e2 ....]

EXAMPLES:     1.) Split method by frequency. Get the values from 10-100 Hz
                  pl = plist('split_type', 'frequencies', ...
                             'frequencies', [10 100]);
                  ao_new = split(a1, pl);

              2.) Split method by time.
                  Get the values from 0.0 to 1.0 Seconds AND from 1.0 to 2.5 seconds
                  pl = plist('split_type', 'times', ...
                             'times',        [0.0 1.0 1.0 2.5]);
                  ao_new = split(a1, pl);

              3.) Split method by samples.
                  Get the samples from 0 to 50 AND from 150 to 200.
                  pl = plist('split_type', 'samples', ...
                             'samples',     [0 50 150 200]);
                  ao_new = split(a1, pl);

              4.1) Select an interval with strings
                   pl = plist('split_type', 'interval', ...
                              'start_time', '14:00:01', ...
                              'end_time',   '14:00:02', ...
                              'format_str', 'HH:MM:SS');
                   ao_new = split(a1, pl);

                   pl = plist('split_type', 'interval', ...
                              'start_time', '14:00:01', ...
                              'duration',   '00:00:02', ...
                              'format_str', 'HH:MM:SS');
                   ao_new = split(a1, pl);

              4.2) Select an interval with time objects
                   pl = plist('split_type', 'interval', ...
                              'start_time', time('14:00:01', 'HH:MM:SS'), ...
                              'end_time',   time('2007-10-10 14:00:03'));
                   ao_new = split(a1, pl);

                   pl = plist('split_type', 'interval', ...
```

```
                              'start_time', time('14:00:01', 'HH:MM:SS'), ...
                              'duration',   time('00:00:05', 'HH:MM:SS'));
                   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', 'HH:MM:
SS'));
                   ao_new = split(a1, pl);

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = split(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: split.m,v 1.34 2008/02/24 13:44:17 hewitson Exp $

 HISTORY: 02-03-07 M Hewitson
            Creation.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- find FIND particular samples that satisfy the input query and return a new AO.
- set SET set an analysis object property.
- split SPLIT split an analysis object into the specified segments.
- times TIMES overloads .* operator for analysis objects.

This function is called by:

- split SPLIT split an analysis object into the specified segments.

SUBFUNCTIONS ^

- function plo = getDefaultPL()
- function bool = isUTCformat(time_string)
- function sec = getSeconds(time_str, format_str)

SOURCE CODE ^

```
function bo = split(varargin)
% SPLIT split an analysis object into the specified segments.
%
%%%
```

```
%
%               The splitting is done as  s<=t<e.
%
%               Arrays of start/stop values should be like: [s1 e1 s2 e2 ....]
%
% EXAMPLES:     1.) Split method by frequency. Get the values from 10-100 Hz
%                   pl = plist('split_type', 'frequencies', ...
%                              'frequencies', [10 100]);
%                   ao_new = split(a1, pl);
%
%               2.) Split method by time.
%                   Get the values from 0.0 to 1.0 Seconds AND from 1.0 to 2.5 seconds
%                   pl = plist('split_type', 'times', ...
%                              'times',        [0.0 1.0 1.0 2.5]);
%                   ao_new = split(a1, pl);
%
%               3.) Split method by samples.
%                   Get the samples from 0 to 50 AND from 150 to 200.
%                   pl = plist('split_type', 'samples', ...
%                              'samples',     [0 50 150 200]);
%                   ao_new = split(a1, pl);
%
%               4.1) Select an interval with strings
%                    pl = plist('split_type', 'interval', ...
%                               'start_time', '14:00:01', ...
%                               'end_time',   '14:00:02', ...
%                               'format_str', 'HH:MM:SS');
%                    ao_new = split(a1, pl);
%
%                    pl = plist('split_type', 'interval', ...
%                               'start_time', '14:00:01', ...
%                               'duration',   '00:00:02', ...
%                               'format_str', 'HH:MM:SS');
%                    ao_new = split(a1, pl);
%
%               4.2) Select an interval with time objects
%                    pl = plist('split_type', 'interval', ...
%                               'start_time', time('14:00:01', 'HH:MM:SS'), ...
%                               'end_time',   time('2007-10-10 14:00:03'));
%                    ao_new = split(a1, pl);
%
%                    pl = plist('split_type', 'interval', ...
%                               'start_time', time('14:00:01', 'HH:MM:SS'), ...
%                               'duration',   time('00:00:05', 'HH:MM:SS'));
%                    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', 'HH:MM:SS'));
%                    ao_new = split(a1, pl);
%
```

```
%               The following call returns a parameter list object that
%               contains the default parameter values:
%               >> pl = split(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: split.m,v 1.34 2008/02/24 13:44:17 hewitson Exp $
%
% HISTORY: 02-03-07 M Hewitson
%            Creation.

%%%
```

```
if ~isempty (pl)
  pl = combine(pl);
end

%% go through analysis objects
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
  d = as(j).data;

  [x,y] = get_xy_values(as(j).data);

  %%%%
```

```
    end

    % go over each interval now
    for i=1:2:ntimes

      is = split_x_axis.value(i);
      ie = split_x_axis.value(i+1);
      idx = find(x>=is & x <ie);

      % create new output data
      d.x = x(idx);
      d.y = y(idx);
      d.name = sprintf('split(%s)', d.name);

      % create new output history
      h = history(ALGONAME, VERSION, plist(param(split_x_axis.type, [is ie])), as(j).hist);
      h.invars = invars(j);

      % Set nsecs for tsdata
      if isa(d, 'tsdata')
        if ~isempty(d.x)
          d.nsecs = d.x(end) - d.x(1) + 1/d.fs;
        else
          d.nsecs = 0;
        end
      end

      % make output analysis object
      b = ao(d, h);

      % set name
      b.name = sprintf('split(%s)', invars{j});

      % Add to output array
      bo = [bo b];

    end

    %%%%
```

```
    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
    disp(sprintf('Split: \f'));
    c = 1;
    dtmp = d;
    dtmp.x = [];
    dtmp.y = [];
    for i=1:2:npairs

      disp(sprintf('\b%03d  ', round(c)));
      if mod(c,11) == 0 && c < npairs
        disp(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
      if isa(d, 'tsdata')
        d.x = x1-x1(1);
      else
        d.x = x1;
      end
      d.y = y1;
      d.name = sprintf('split(%s)[%d]', dtmp.name, c);

      % set t0,nsecs if this is tsdata
      if isa(d, 'tsdata')
        t0 = dtmp.t0 + x1(1);
        d.t0  = t0;
        d.nsecs = length(y1) ./ dtmp.fs;
      end

      % create new output history
      h = history(ALGONAME, VERSION, plist(param('samples', [is ie])), as(j).hist);
```

```
      h.invars = invars;

      % make output analysis object
      b = ao(d, h);

      % set name
      b.name = sprintf('split(%s)[%d]', invars{j}, c);

      % Add to output array
      bo = [bo b];
      c = c + 1;
    end

    %%%%
```

```
    end_time   = find(pl, 'end_time');
    format_str = find(pl, 'format_str');
    timespan   = find(pl, 'timespan');

    %%%%%%%%%%%%%%%%%%%%%%%%   Convert the time object   %%%%%%%%%%%%%%%%%%%%%%%%%
    if ~isempty(start_time) && isa(start_time, 'time')
      if ~isempty(format_str)
        start_time = set(start_time, 'timeformat', format_str);
      else
        format_str = char(start_time.timeformat);
      end
      %     format_str = char(start_time.timeformat);
      start_time = char(start_time);
    end

    if ~isempty(end_time) && isa(end_time, 'time')
      if ~isempty(format_str)
        end_time = set(end_time, 'timeformat', format_str);
      else
        format_str = char(end_time.timeformat);
      end
      %     format_str = char(end_time.timeformat);
      end_time   = char(end_time);
    end

    if ~isempty(duration) && isa(duration, 'time')
      if ~isempty(format_str)
        duration = set(duration, 'timeformat', format_str);
      else
        format_str = char(duration.timeformat);
      end
      %     format_str = char(end_time.timeformat);
      duration   = char(duration);
    end

    %%%%%%%%%%%%%%%%%%%%%%   Convert the time span object   %%%%%%%%%%%%%%%%%%%%%%%
    if ~isempty(timespan)
      start_time = char(timespan.start);
      end_time   = char(timespan.end);
      format_str = char(timespan.timeformat);
      duration   = '';
    end

    UTC = false;

    %% check the input values
    if ~isa(as(j).data, 'tsdata')
      error ('### the split type: ''duration'' only works with tsdata.')
    end
```

```

    if (isempty(start_time) && ...
        (isempty(duration) || isempty(end_time)))
      error ('### please specify the key ''start_time'' AND ''end_stime'' OR ''duration'' in the parameter list')
    end

    if (ischar(start_time) ||     ...
        ischar(end_time)   ||     ...
        ischar(duration)     ) && ...
        isempty(format_str)
      %% are the time strings in UTC (yyy-mm-dd HH:MM:SS)
      %% then set the format string to the UTC format.
      if isUTCformat(start_time) || ...
          isUTCformat(end_time)   || ...
          isUTCformat(duration)
        format_str = 'yyyy-mm-dd HH:MM:SS';
      else
        error ([char (10) '### please specify the key ''format_str'' for the keys ' ...
          '''start_time'', ''end_time'' OR/AND ''duration'''])
      end
    end

    disp('* splitting by interval ...');

    %% convert the start time
    if isUTCformat(start_time)
      UTC = true;
      start_time = time(start_time);
      start_time = start_time.utc_epoch_milli/1000;
    elseif ischar (start_time)
      start_time = getSeconds(start_time, format_str);
    end

    %% convert the end time
    if isUTCformat(end_time)
      UTC = true;
      end_time = time(end_time);
      end_time = end_time.utc_epoch_milli/1000;
    elseif ischar (end_time)
      end_time = getSeconds(end_time, format_str);
    end

    %% convert the duration
    if ~isempty(end_time) && ~isempty(duration)
      warning (['### in the parameter list are both parameter '...
        '''end_time'' and ''duration''. Using the parameter ''end_time'''])
    elseif ischar(duration) && ~isempty(duration)
      duration = getSeconds(duration, format_str);
    end

```

```
    %% get/convert the t0
    t0 = as(j).data.t0;
    t_unix_time = t0.utc_epoch_milli/1000;
    t1 = x(1);

    if UTC == true
      ts = start_time-t_unix_time+t1;
      if ~isempty(end_time)
        te = end_time-t_unix_time+t1;
      else
        te = ts+duration;
      end
    else

      t0_offset = format(t0, format_str);
      t0_offset = getSeconds(t0_offset, format_str);

      ts = start_time+t1-t0_offset;
      if ~isempty(end_time)
        te = end_time+t1-t0_offset;
      else
        te = start_time+duration+t1-t0_offset;
      end
    end

    idx = find(x>=ts & x <te);

    % create new output data
    d.x = x(idx);
    d.y = y(idx);
    d.name = sprintf('split(%s)', d.name);

    % Set nsecs for tsdata
    if isa(d, 'tsdata')
      if ~isempty(d.x)
        d.nsecs = d.x(end) - d.x(1) + 1/d.fs;
      else
        d.nsecs = 0;
      end
    end

    % create new output history
    h = history(ALGONAME, VERSION, pl, as(j).hist);
    h.invars = invars;

    % make output analysis object
    b = ao(d, h);

    % set name
    b.name = sprintf('split(%s)', invars{j});

    % Add to output array
```

```
    bo = [bo b];


  else
    error('### do not know how to split.')
  end

end % for j = 1:numel(as)

% Reshape the ouput to the same size of the input
if numel(bo) == numel(as)
  bo = reshape(bo, size(as));
end

end % split

%%%
```

```
function bool = isUTCformat(time_string)

  bool = false;

  if ischar(time_string)
    if regexp(time_string, '\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}')
      bool = true;
    end
  end

end % isUTCformat


%%%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

sqrt

PURPOSE ^

SQRT overloads the sqrt operator for Analysis objects. Square root.

SYNOPSIS ^

function ao_out = sqrt (varargin)

DESCRIPTION ^

```
 SQRT overloads the sqrt operator for Analysis objects. Square root.

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

 DESCRIPTION: SQRT overloads the sqrt operator for Analysis objects.
              Square root.
              SQRT(ao) is the square root of the elements of ao.data.

 CALL: ao_out = sqrt(ao_in);
       ao_out = sqrt(ao_in, pl);
       ao_out = sqrt(ao1, pl1, ao_vector, ao_matrix, pl2);

 POSSIBLE VALUES: ao_in  = [ao2 ao3]
                  ao_in  = ao_vector
                  ao_in  = ao_matrix

 PARAMETER LIST:  <key>           <value>             <description>
                          tsdata  fsdata  xydata
                 'xdata'   't'     'f'      'x'     compute the xdata
                 'ydata'   'x'     'xx'     'y'     compute the ydata

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = sqrt(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = sqrt(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = sqrt(ao,'Category')
```

```
 HISTORY: 24-05-2007 Diepholz
             Creation

%%%
```

```
% The following call returns a string that contains the routine CVS version:
%
% >> version = sqrt(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = sqrt(ao,'Category')
%
% HISTORY: 24-05-2007 Diepholz
%             Creation
%
%%%
```

```
0080
0081   %% Add the history from the ao object to the history
0082   h = set(h, 'inhists', [a.hist]);
0083
0084   %% Set the var_name to the history
0085   h = set(h, 'invars', cellstr(invars{j}));
0086
0087   %% create a new analysis objects
0088   new_ao = ao  (sqrt_data, h);
0089   new_ao = setnh(new_ao, 'name', sprintf('sqrt(%s)',invars{j}) );
0090
0091   ao_out = [ao_out new_ao];
0092
0093 end
0094
0095 % Reshape the ouput to the same size of the input
0096 ao_out = reshape(ao_out, size(ao_set));
0097
0098 %% Get default params
0099 function pl_default = getDefaultPL()
0100
0101   pl_default = plist([param('xdata',  '')
0102                       param('ydata',  '')]);
0103
0104 % END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

std

PURPOSE ^

STD overloads the std operator for Analysis objects. Compute the standard deviation.

SYNOPSIS ^

function ao_out = std (varargin)

DESCRIPTION ^

```
 STD overloads the std operator for Analysis objects. Compute the standard deviation.

%%%
```

```
 The following call returns a string that contains the routine category:

 >> category = std(ao,'Category')

 HISTORY: 23-05-2007 Diepholz
             Creation

%%%
```

```
%                  'dim'          1 or 2 or 3 ...    takes the std along the
%                                                    dimension dim
%                         tsdata  fsdata  xydata
%                'xdata'   't'     'f'      'x'     compute the std of the x-axis
%                'ydata'   'x'     'xx'     'y'     compute the std of the y-axis
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = std(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = std(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = std(ao,'Category')
%
% HISTORY: 23-05-2007 Diepholz
%             Creation
%
%%%
```

```
  in_names{end+1} = inputname(ii);

  if isnumeric(varargin{ii})
    ps = [ps plist(param('dim', varargin{ii}))];
  end
end

[ao_set, pl, invars] = collect_inputs(varargin, in_names);

pl = combine(pl, ps);

%% go through analysis objects
for j=1:numel(ao_set)
  a = ao_set(j);

  [h, std_data] = single_operation(a.data, 'std', pl);

  %% Add the history from the ao object to the history
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  % convert to cdata type
  std_x    = [];
  std_y    = [];

  do_xdata = find(pl, 'xdata');
  do_ydata = find(pl, 'ydata');

  %% Is no axis entry in the parameter list
  %% then set the default axis = xdata
  if isempty(do_xdata) && isempty(do_ydata)
    do_ydata = 'yes';
  end

  %% Get the operation result. It is stored in the individual axis
  pl_get_axis = plist([param('xdata', do_xdata) param('ydata', do_ydata)]);
  [std_x, std_y] = get_xy_values(std_data, pl_get_axis);

  if ~isempty(do_xdata) && ~isempty(do_ydata)
    std_cdata = cdata([std_x std_y]);
  elseif ~isempty(do_xdata)
    std_cdata = cdata(std_x);
  else
    std_cdata = cdata(std_y);
  end

  std_cdata = set (std_cdata, 'name', std_data.name);
  std_cdata = set (std_cdata, 'xunits', std_data.xunits);
  std_cdata = set (std_cdata, 'yunits', std_data.yunits);

```

```
  %% create a new analysis objects
  new_ao = ao  (std_cdata, h);
  new_ao = setnh(new_ao, 'name', sprintf('std(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('dim',    '')
                      param('W',      [])
                      param('xdata',  '')
                      param('ydata',  '')]);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsasgn

PURPOSE ^

SUBSASGN define index assignment for analysis object properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for analysis object properties.

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

 DESCRIPTION: SUBSASGN define index assignment for analysis object properties.

 EXAMPLES:    All possible assignments are possible.

 VERSION:     $Id: subsasgn.m,v 1.9 2008/02/25 18:06:31 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for analysis object properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSASGN define index assignment for analysis object properties.
```

```
%
% EXAMPLES:    All possible assignments are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.9 2008/02/25 18:06:31 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

sum

PURPOSE ^

SUM overloads the sum operator for Analysis objects. Compute the sum value.

SYNOPSIS ^

function ao_out = sum (varargin)

DESCRIPTION ^

```
 SUM overloads the sum operator for Analysis objects. Compute the sum value.

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

 DESCRIPTION: SUM overloads the sum operator for Analysis objects.
              Compute the sum value.
              SUM(ao) is the sum value of the elements in ao.data.

 CALL: ao_out = sum(ao_in);
       ao_out = sum(ao_in, dim);
       ao_out = sum(ao_in, pl);
       ao_out = sum(ao1, pl1, ao_vector, ao_matrix, pl2);

 POSSIBLE VALUES: ao_in  = [ao2 ao3]
                  ao_in  = ao_vector
                  ao_in  = ao_matrix

 PARAMETER LIST:  <key>           <value>             <description>
                  'dim'          1 or 2 or 3 ...    takes the sum along the
                                                     dimension dim
                          tsdata  fsdata  xydata
                 'xdata'   't'     'f'      'x'     compute the sum of the x-axis
                 'ydata'   'x'     'xx'     'y'     compute the sum of the y-axis

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = sum(ao,'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = sum(ao,'Version')

 The following call returns a string that contains the routine category:
```

```
>> category = sum(ao,'Category')

HISTORY: 25-05-2007 Diepholz
            Creation

%%%
```

```
%                           tsdata  fsdata  xydata
%                 'xdata'   't'     'f'      'x'     compute the sum of the x-axis
%                 'ydata'   'x'     'xx'     'y'     compute the sum of the y-axis
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = sum(ao,'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = sum(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = sum(ao,'Category')
%
% HISTORY: 25-05-2007 Diepholz
%             Creation
%
%%%
```

```
  if isnumeric(varargin{ii})
    ps = [ps plist(param('dim', varargin{ii}))];
  end
end

[ao_set, pl, invars] = collect_inputs(varargin, in_names);

pl = combine(pl, ps);

%% go through analysis objects
for j=1:numel(ao_set)
  a = ao_set(j);

  [h, sum_data] = single_operation(a.data, 'sum', pl);

  %% Add the history from the ao object to the history
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  do_xdata = find(pl, 'xdata');
  do_ydata = find(pl, 'ydata');

  %% Is no axis entry in the parameter list
  %% then set the default axis = xdata
  if isempty(do_xdata) && isempty(do_ydata)
    do_ydata = 'yes';
  end

  %% Get the operation result. It is stored in the individual axis
  pl_get_axis = plist([param('xdata', do_xdata) param('ydata', do_ydata)]);
  [sum_x, sum_y] = get_xy_values(sum_data, pl_get_axis);

  if ~isempty(do_xdata) && ~isempty(do_ydata)
    sum_cdata = cdata([sum_x sum_y]);
  elseif ~isempty(do_xdata)
    sum_cdata = cdata(sum_x);
  else
    sum_cdata = cdata(sum_y);
  end

  sum_cdata = set (sum_cdata, 'name', sum_data.name);
  sum_cdata = set (sum_cdata, 'xunits', sum_data.xunits);
  sum_cdata = set (sum_cdata, 'yunits', sum_data.yunits);

  %% create a new analysis objects
  new_ao = a;
  new_ao = ao  (sum_cdata, h);
  new_ao = setnh(new_ao, 'name', sprintf('sum(%s)',invars{j}) );

```

```
  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));

%% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('xdata', ''));
plo = append(plo, param('ydata', ''));
plo = append(plo, param('dim',   []));


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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)
             [u, s]    = svd (a,pl)
             [u, s, v] = svd (a,pl)
             [u, s, v] = svd (a)

 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 produve economy size decomposition
                       - options are the same as for the matlab function

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = svd(ao,'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = svd(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = svd(ao,'Category')

 VERSION:     $Id: svd.m,v 1.15 2008/02/12 10:11:13 mauro Exp $
```

```
 HISTORY:     08-05-07A Monsky
                Creation.

%%%
```

```
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = svd(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = svd(ao,'Category')
%
% VERSION:     $Id: svd.m,v 1.15 2008/02/12 10:11:13 mauro Exp $
%
% HISTORY:     08-05-07A Monsky
%                Creation.
%
%%%
```

```

% check plist
if ~isempty (pl)
  pl = combine(pl);
end

% Check input analysis object
for j=1:numel(as)
  a = as(j);

  d = get(a, 'data');
  dinfo = whos('d');

  % Which data type do we have
  dtype = dinfo.class;

  udata = [];
  sdata = [];
  vdata = [];

  switch dtype
    case 'cdata'
      if nargout <= 1
        [h, udata] = single_operation(a.data, 'svd',pl);
      elseif nargout == 2
        [h, udata, sdata] = single_operation(a.data, 'svd',pl);
      elseif nargout == 3
        [h, udata, sdata, vdata] = single_operation(a.data, 'svd',pl);
      end
      %create analysis object(s)
      h = set(h, 'inhists', [a.hist]);

      %% Set the var_name to the history
      h = set(h, 'invars', cellstr(invars{j}));

      u = ao(udata, h);
      u = setnh(u, 'name',  sprintf('u_svd(%s)', char(invars{j})));
      U = [U u];
      if ~isempty(sdata)
        s = ao(sdata, h);
        s = setnh(s, 'name',  sprintf('s_svd(%s)', char(invars{j})));
        S = [S s];
      end
      if ~isempty(vdata)
        v = ao(vdata, h);
        v = setnh(v, 'name',  sprintf('v_svd(%s)', char(invars{j})));
        V = [V v];
      end

    case {'tsdata','fsdata','xydata'}
      error('### this function works for cdata type AO only')
```

```
    otherwise
      error('### unknown data type.')
  end

end

% Reshape the ouput to the same size of the input
U = reshape(U, size(as));
varargout{1} = U;

if nargout > 1
  % Reshape the ouput to the same size of the input
  S = reshape(S, size(as));
  varargout{2} = S;
  if nargout > 2
    % Reshape the ouput to the same size of the input
    V = reshape(V, size(as));
    varargout{3} = V;
    if nargout > 3
      error('### wrong number of outputs')
    end
  end
end

%% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, 'option', []);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

tan

PURPOSE ^

TAN overloads the tan operator for Analysis objects. Tangent of argument in radians.

SYNOPSIS ^

function ao_out = tan (varargin)

DESCRIPTION ^

```
 TAN overloads the tan operator for Analysis objects. Tangent of argument in radians.

%%%
```

```
              Creation

%%%
```

```
%
% >> version = tan(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = tan(ao,'Category')
%
% HISTORY: 07-05-2007 Diepholz
%             Creation
%
%%%
```

```
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  %% create a new analysis objects
  new_ao = a;
  new_ao = ao  (tan_data, h);
  new_ao = setnh(new_ao, 'name', sprintf('tan(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('xdata',  '')
                      param('ydata',  '')]);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

times

PURPOSE ^

TIMES overloads .* operator for analysis objects.

SYNOPSIS ^

function b = times(varargin)

DESCRIPTION ^

```
 TIMES overloads .* operator for analysis objects.

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

 DESCRIPTION: TIMES overloads .* operator for analysis objects.
              This just does the same as times (.*) for analysis objects.

 CALL:        a = a1.*5;

 VERSION:     $Id: times.m,v 1.13 2008/02/13 18:35:22 hewitson Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = times(ao,'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = times(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = times(ao,'Category')

 HISTORY: 05-02-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

- split SPLIT split an analysis object into the specified segments.

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function b = times(varargin)
% TIMES overloads .* operator for analysis objects.
%
%%%
```

```
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      b = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      b = VERSION;
      return
    elseif strcmp(in, 'Category')
      b = CATEGORY;
      return
    end
  end
end

% capture input variable names
invars = {};
for j=1:nargin
  iname = inputname(j);
  if isempty(iname) && isnumeric(varargin{j})
    iname = num2str(varargin{j});
  elseif isempty(iname) && ischar(varargin{j})
    iname = varargin{j};
  end
  invars = [invars cellstr(iname)];
end

op = '.*';

[a1,a2,do] = aooperate(varargin, op);

%-------- create output AO

% make a new history object
h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
h = set(h, 'invars', invars);

% get names for output
if isempty(char(invars{1}))
  n1 = a1.name;
else
  n1 = char(invars{1});
end
if isempty(char(invars{2}))
  n2 = a2.name;
else
  n2 = char(invars{2});
end

% make output analysis object
```

```
% op = op(2:end);
b = ao(do, h);
b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));


% Get default params
function plo = getDefaultPL()

plo = plist();


% END
```

Generated on Tue 26-Feb-2008 10:52:52 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.

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

 DESCRIPTION: TRANSPOSE overloads the .' operator for Analysis Objects.

 CALL:        a = a1.'     % only with data = cdata

 VERSION:     $Id: transpose.m,v 1.14 2008/02/12 10:11:13 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = transpose(ao,'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = transpose(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = transpose(ao,'Category')


 HISTORY: 08-05-07 A Monsky
            Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- get GET get ao properties.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = transpose(varargin)
% TRANSPOSE overloads the .' operator for Analysis Objects.
%
%%%
```

```

if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    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
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, ps, invars] = collect_inputs(varargin, in_names);

% check plist
if isempty(ps)
  pl = getDefaultPL();
else
  pl = combine(ps, getDefaultPL);
end

%create empty plist
plo = plist();

%% go through analysis objects
for j=1:numel(as)
  a = as(j);
  d = get(a, 'data');
  dinfo = whos('d');

  % Which data type do we have
  dtype = dinfo.class;

  switch dtype
    case 'cdata'
      %disp('* TRANSPOSE of cdata object');
      % make a new cdata object
      [h, c] = single_operation(d, 'transpose', pl);

      % make output analysis object
```

```
      h = set(h, 'inhists', [a.hist]);

      %% Set the var_name to the history
      h = set(h, 'invars', cellstr(invars{j}));

      b = ao(c, h);
      b = setnh(b, 'name',  sprintf('transpose(%s)', char(invars{j})));

    case {'tsdata','fsdata','xydata'}
      error('### this function works for cdata type AO only')
    otherwise
      error('### unknown data type.')

  end

  % 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;

%% Get default params
function plo = getDefaultPL()

plo = plist();

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

upsample

PURPOSE ^

UPSAMPLE overloads upsample function for AOs.

SYNOPSIS ^

function bs = upsample(varargin)

DESCRIPTION ^

```
 UPSAMPLE overloads upsample function for AOs.

%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

- ao AO analysis object class constructor.
- char CHAR overloads char() function for analysis objects.
- find FIND particular samples that satisfy the input query and return a new AO.
- phase PHASE overloads the ltpda_phase operator for Analysis objects.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.
- upsample UPSAMPLE overloads upsample function for AOs.

This function is called by:

- upsample UPSAMPLE overloads upsample function for AOs.

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function bs = upsample(varargin)
% UPSAMPLE overloads upsample function for AOs.
%
%%%
```

```
% The following call returns a string that contains the routine category:
%
% >> category = upsample(ao,'Category')
%
% HISTORY: 19-03-07 M Hewitson
%             Creation
%
%%%
```

```
      if numel(varargin{j}) == 1
        invars{end+1} = ao_name;
      else
        for ii=1:numel(varargin{j})
          [I,J] = ind2sub(size(varargin{j}),ii);
          invars{end+1} = sprintf('%s(%d,%d)', ao_name, I, J);
        end
      end
    else
      warning('!!! ignoring non-time-series AO.');
    end
  end
  if isa(a, 'plist')
    pl = a;
  end
end


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

% 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

for j=1:numel(as)

  d  = as(j).data;

  % upsample y
  y = upsample(d.y, floor(Nup), phase);

  % make new output tsdata
  do = tsdata(y, Nup*d.fs);
  d = set(d, 'x', do.x);
  d = set(d, 'y', do.y);
  d = set(d, 'fs', do.fs);

  %--------- create output AO

  % make a new history object
  h = history(ALGONAME, VERSION, pl, as(j).hist);
  h = set(h, 'invars', invars);

```

```
  % make output analysis object
  b = ao(d, h);
  b = setnh(b, 'name', sprintf('upsample(%s)', invars{j}));

  bs = [bs b];

end

% Reshape the ouput to the same size of the input
bs = reshape(bs, size(as));

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('N', 1));
plo = append(plo, param('phase', 0));

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

var

PURPOSE ^

VAR overloads the var operator for Analysis objects. Compute the variance.

SYNOPSIS ^

function ao_out = var (varargin)

DESCRIPTION ^

```
 VAR overloads the var operator for Analysis objects. Compute the variance.

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

 DESCRIPTION: VAR overloads the var operator for Analysis objects.
              Compute the variance.
              VAR(ao) is the variance of the values in ao.data.

 CALL: ao_out = var(ao_in);
       ao_out = var(ao_in, dim);
       ao_out = var(ao_in, W, DIM); --> use the parameterlist
       ao_out = var(ao_in, pl);
       ao_out = var(ao1, pl1, ao_vector, ao_matrix, pl2);

 POSSIBLE VALUES: ao_in  = [ao2 ao3]
                  ao_in  = ao_vector
                  ao_in  = ao_matrix

 PARAMETER LIST:  <key>           <value>             <description>
                   'W'               []             represent the weight vector W
                  'dim'          1 or 2 or 3 ...    takes the var along the
                                                    dimension dim
                          tsdata  fsdata  xydata
                 'xdata'   't'     'f'      'x'     compute the var of the x-axis
                 'ydata'   'x'     'xx'     'y'     compute the var of the y-axis

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = var(ao, 'Params')

 HISTORY: 23-05-2007 Diepholz
             Creation
```

```
%%%
```

```
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = var(ao, 'Params')
%
% HISTORY: 23-05-2007 Diepholz
%             Creation
%
%%%
```

```
  [h, var_data] = single_operation(a.data, 'var', pl);

  %% Add the history from the ao object to the history
  h = set(h, 'inhists', [a.hist]);

  %% Set the var_name to the history
  h = set(h, 'invars', cellstr(invars{j}));

  do_xdata = find(pl, 'xdata');
  do_ydata = find(pl, 'ydata');

  %% Is no axis entry in the parameter list
  %% then set the default axis = xdata
  if isempty(do_xdata) && isempty(do_ydata)
    do_ydata = 'yes';
  end

  %% Get the operation result. It is stored in the individual axis
  pl_get_axis = plist([param('xdata', do_xdata) param('ydata', do_ydata)]);
  [var_x, var_y] = get_xy_values(var_data, pl_get_axis);

  if ~isempty(do_xdata) && ~isempty(do_ydata)
    var_cdata = cdata([var_x var_y]);
  elseif ~isempty(do_xdata)
    var_cdata = cdata(var_x);
  else
    var_cdata = cdata(var_y);
  end

  var_cdata = set (var_cdata, 'name', var_data.name);
  var_cdata = set (var_cdata, 'xunits', var_data.xunits);
  var_cdata = set (var_cdata, 'yunits', var_data.yunits);

  %% create a new analysis objects
  new_ao = ao  (var_cdata, h);
  new_ao = setnh(new_ao, 'name', sprintf('var(%s)',invars{j}) );

  ao_out = [ao_out new_ao];

end

% Reshape the ouput to the same size of the input
ao_out = reshape(ao_out, size(ao_set));


%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist([param('dim',    '')
                      param('W',      [])
```

```
                        param('xdata',  '')
                        param('ydata',  '')]);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 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 parameter to the parameter list.
char	CHAR convert a parameter list into a string.
combine	COMBINE multiple parameter lists (plist objects) into a single plist.
display	DISPLAY display plist object.
eq	EQ overloads the == operator for plist objects.
find	FIND overloads find routine for a parameter list.
get	GET get parameter list properties.
isfield	ISFIELD tests if the given field is one of the object properties.
isparam	ISPARAM look for a given key in the parameter list.
isvalid	ISVALID tests if the given plist has all the correct fields of the correct type.
ne	NE overloads the ~= operator for param objects.
nparams	NPARAMS returns the number of param objects in the list.
plist	PLIST Plist class object constructor.
pset	PSET set parameter list properties.
pzmresp	PZMRESP shadow function for the pzmodel/resp.
remove	REMOVE remove a parameter from the parameter list.
resp	RESP shadows miir/iirResp.
save	SAVE a plist object to file.
set	SET set parameter list properties.
string	STRING converts a plist object to a command string which will recreate the plist object.
subsasgn	SUBSASGN define index assignment for parameter list properties.
subsref	SUBSREF Define field name indexing for parameter list objects.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

char

PURPOSE ^

CHAR convert a parameter list into a string.

SYNOPSIS ^

function pstr = char(pl, varargin)

DESCRIPTION ^

```
 CHAR convert a parameter list into a string.

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

 DESCRIPTION: CHAR convert a parameter list into a string.

 CALL:        string = char(plist);

 VERSION:     $Id: char.m,v 1.13 2008/02/14 08:29:09 mauro Exp $

 HISTORY:     04-02-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a parameter list into a string.
- get GET get parameter list properties.
- plist PLIST Plist class object constructor.

This function is called by:

- append APPEND append a parameter to the parameter list.
- char CHAR convert a parameter list into a string.
- combine COMBINE multiple parameter lists (plist objects) into a single plist.
- display DISPLAY display plist object.
- find FIND overloads find routine for a parameter list.
- ne NE overloads the ~= operator for param objects.
- nparams NPARAMS returns the number of param objects in the list.

- plist PLIST Plist class object constructor.
- pset PSET set parameter list properties.
- pzmresp PZMRESP shadow function for the pzmodel/resp.
- resp RESP shadows miir/iirResp.
- save SAVE a plist object to file.
- string STRING converts a plist object to a command string which will recreate the plist object.
- subsasgn SUBSASGN define index assignment for parameter list properties.
- subsref SUBSREF Define field name indexing for parameter list objects.

SOURCE CODE ^

```
function pstr = char(pl, varargin)
% CHAR convert a parameter list into a string.
%
%%%
```

```
pstr = ['('];
for j=1:np
  p = pl.params(j);

  %%%%%%%%%%   cell   %%%%%%%%%%
  if iscell(p.val)
    pstr = [pstr p.key '={'];
    for ii = 1:length(p.val)
      if ischar(p.val{ii})
        pstr = [pstr '''' p.val{ii} ''', ' ];
      elseif isnumeric(p.val{ii})
        pstr = [pstr '' ltpda_mat2str(p.val{ii}) ', '];
      else
        pstr = [pstr '' class(p.val{ii}) '-object, '];
      end
    end
    if strcmp(pstr(end-1:end), ', ')
      pstr = [pstr(1:end-2) '}, '];
    else
      pstr = [pstr '}, '];
    end

  %%%%%%%%%%   character   %%%%%%%%%%
  elseif ischar(p.val)
    sval = char(p.val);
    if length(sval) == 0
      sval = [''' '''];
    elseif length(sval) >= 10
      sval = [sval(1:10) '...'];
    end
    pstr = [pstr p.key '=' sval ', '];

  %%%%%%%%%%   numbers   %%%%%%%%%%
  elseif isnumeric(p.val)
    val_str = ltpda_mat2str(p.val);
    if length(val_str) > 25
      val_str = [val_str(1:25) ' ...]'];
    end
    pstr = [pstr p.key '=' val_str ', '];

  %%%%%%%%%%   specwin   %%%%%%%%%%
  elseif isa(p.val, 'specwin')
    % display only the window name here
    w = p.val;
    pstr = [pstr p.key '=' w.name ', '];

  %%%%%%%%%%   miir   %%%%%%%%%%
  elseif isa(p.val, 'miir')
    f = p.val;
    if strcmp(get(f, 'infile'), '')
      pstr = [pstr p.key '=' get(f, 'name') ', '];
    else
```

```
        pstr = [pstr p.key '=' get(f, 'infile') ', '];
      end

    %%%%%%%%%%   mfir   %%%%%%%%%%
    elseif isa(p.val, 'mfir')
      f = p.val;
      pstr = [pstr p.key '=' get(f, 'name') ', '];

    %%%%%%%%%%   time, timeformat, timespan   %%%%%%%%%%
    elseif isa(p.val, 'time') || isa(p.val, 'timeformat') || isa(p.val, 'timespan')
      pstr = [pstr p.key '=' char(p.val) ', '];

    %%%%%%%%%%   all other classes   %%%%%%%%%%
    else
      pstr = [pstr p.key '=' class(p.val) '-object, '];
%     else
%       warning('!!! unknown parameter type. Can''t convert to string.');
    end
  end
  % remove last ', '
  if length(pstr)>1
    pstr = pstr(1:end-2);
  end
  % close bracket
  pstr = [pstr ')'];
end


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

combine

PURPOSE ^

COMBINE multiple parameter lists (plist objects) into a single plist.

SYNOPSIS ^

function plo = 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(param('A', 1));
              >> pl2 = plist(param('A', 3));
              >> plo = plist(pl1, pl2)

              Then plo will contain a parameter 'A' with value 1.

 VERSION:     $Id: combine.m,v 1.8 2008/02/14 08:29:09 mauro Exp $

 HISTORY:     28-04-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a parameter list into a string.
- nparams NPARAMS returns the number of param objects in the list.
- plist PLIST Plist class object constructor.

This function is called by:

- plist PLIST Plist class object constructor.

SUBFUNCTIONS ^

- function pl = add_param(pl, param)

SOURCE CODE ^

```
function plo = combine(varargin)
% COMBINE multiple parameter lists (plist objects) into a single plist.
%
%%%
```

```
    end
  end
end

% get input plists
% this code allows for inputs of the form:
% >> pl = combine([p1 p2], p, [p3 p4], p5, p6)
% or some combination of that.
pls = [];
for j=1:length(varargin)
  pj = varargin{j};
  if isa(pj, 'plist')
    pj = reshape(pj, 1, []);
    pls = [pls pj];
  end
end

% The properties of the output plist is corresponding to the input plist
plo = pls(1);

% Loop over all plist's
for ii = 2:length(pls)

  % Loop over all params in the current plist
  for jj = 1:length(pls(ii).params)

    ps = pls(ii).params(jj);
    plo = add_param(plo, ps);

  end

end


%%%
```

```
pl.params = [pl.params param];


% END

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

find

PURPOSE ^

FIND overloads find routine for a parameter list.

SYNOPSIS ^

function a = find(pl, key)

DESCRIPTION ^

```
 FIND overloads find routine for a parameter list.

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

 DESCRIPTION: FIND overloads find routine for a parameter list.
              Returns the value corresponding to the first parameters in
              the list with key 'key'.

 CALL:        a = find(pl, 'key')

 VERSION:     $Id: find.m,v 1.6 2008/02/14 08:29:09 mauro Exp $

 HISTORY:     02-02-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a parameter list into a string.
- get GET get parameter list properties.
- plist PLIST Plist class object constructor.

This function is called by:

- plist PLIST Plist class object constructor.
- pzmresp PZMRESP shadow function for the pzmodel/resp.
- resp RESP shadows miir/iirResp.

SOURCE CODE ^

```
function a = find(pl, key)
% FIND overloads find routine for a parameter list.
%
%%%
```

```
0052
0053
0054 % END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

remove

PURPOSE ^

REMOVE remove a parameter from the parameter list.

SYNOPSIS ^

function pl = remove(pl, val)

DESCRIPTION ^

```
 REMOVE remove a parameter from the parameter list.

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

 DESCRIPTION: REMOVE remove a parameter from the parameter list.

 CALL:        pl = remove(pl, 2)      - removes 2nd in the list
              pl = remove(pl, 'key')  - removes parameter called 'key'

 VERSION:     $Id: remove.m,v 1.5 2008/02/14 08:29:09 mauro Exp $

 HISTORY:     30-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- get GET get parameter list properties.
- nparams NPARAMS returns the number of param objects in the list.
- plist PLIST Plist class object constructor.

This function is called by:

- plist PLIST Plist class object constructor.

SOURCE CODE ^

```
function pl = remove(pl, val)
% REMOVE remove a parameter from the parameter list.
```

```
%
%%%
```

```
  idx = [1:val-1 val+1:np];
  if max(idx)>np
    error('### index exceeds number of parameters');
  end
  pl = plist(params(idx));

%%%%%   Unknown method   %%%%%
else
  error('### unknown indexing method')
end



% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

append

PURPOSE ^

APPEND append a parameter to the parameter list.

SYNOPSIS ^

function pl = append(pl, varargin)

DESCRIPTION ^

```
 APPEND append a parameter to the parameter list.

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

 DESCRIPTION: APPEND append a parameter to the parameter list.

 CALL:        pl = append(pl,p);
              pl = append(pl,pl);
              pl = append(pl, 'key1', 'value1')

 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.

 VERSION:     $Id: append.m,v 1.8 2008/02/14 08:29:09 mauro Exp $

 HISTORY:     30-01-07 M Hewitson
                Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a parameter list into a string.
- nparams NPARAMS returns the number of param objects in the list.
- plist PLIST Plist class object constructor.

This function is called by:

- pset PSET set parameter list properties.
- save SAVE a plist object to file.

SUBFUNCTIONS ^

- function pl = add_param(pl, param)

SOURCE CODE ^

```
function pl = append(pl, varargin)
% APPEND append a parameter to the parameter list.
%
%%%
```

```
  if isa(argin{1}, 'param')
    pp = argin{1};
    pl = add_param(pl, pp);
    argin = argin(2:end);

  elseif isa(argin{1}, 'plist')
    pl_in = argin{1};

    for ii = 1:nparams(pl_in)
      pl = add_param(pl, pl_in.params(ii));
    end
    argin = argin(2:end);

  else
    if length(argin) >= 2
      key = argin{1};
      val = argin{2};
      pl = add_param(pl, param(key, val));
      argin = argin(3:end);
    else
      error('### There is a key [%s] without a value left.', key);
      argin = argin(2:end);
    end
  end
end


%%%
```

```
0095
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY display plist object.

SYNOPSIS ^

function txt = display(pls, varargin)

DESCRIPTION ^

```
 DISPLAY display plist object.

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

 DESCRIPTION: DISPLAY display plist object.

 VERSION: $Id: display.m,v 1.5 2008/02/14 08:29:09 mauro Exp $

 HISTORY: 30-01-2007 M Hewitson
            Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a parameter list into a string.
- display DISPLAY display plist object.
- get GET get parameter list properties.
- nparams NPARAMS returns the number of param objects in the list.
- plist PLIST Plist class object constructor.

This function is called by:

- display DISPLAY display plist object.
- set SET set parameter list properties.

SUBFUNCTIONS ^

- function new_txt = single_cell(txt_field)

SOURCE CODE ^

```
function txt = display(pls, varargin)
% DISPLAY display plist object.
%
%%%
```

```

  txt = single_cell(txt);

  banner_end(1:length(banner_start)) = '-';
  txt{end+1} = banner_end;

end

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end


%%%
```

eq

PURPOSE ^

EQ overloads the == operator for plist objects.

SYNOPSIS ^

function result = eq(c1, c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for plist objects.

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

 DESCRIPTION: EQ overloads the == operator for plist objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input plist objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two plist objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.9 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```

VERSION       = '$Id: eq.m,v 1.9 2008/02/18 14:41:13 ingo Exp $';
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get parameter list properties.

SYNOPSIS ^

function val = get(pl, propName)

DESCRIPTION ^

```
 GET get parameter list properties.

%%%
```

```
function val = get(pl, propName)
% GET get parameter list properties.
%
%%%
```

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(plist, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(plist, 'Version')

              The following call returns the category:

              >> category = isfield(plist, 'Category')

 VERSION:     $Id: isfield.m,v 1.4 2008/02/14 08:29:09 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- plist PLIST Plist class object constructor.

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)
% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
for j=1:nargin
  if isa(varargin{j}, 'plist')
    as = [as varargin{j}];
  end
  if ischar(varargin{j})
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isparam

PURPOSE ^

ISPARAM look for a given key in the parameter list.

SYNOPSIS ^

function res = isparam(pl, key)

DESCRIPTION ^

```
 ISPARAM look for a given key in the parameter list.

%%%
```

```
%
% CALL:        res = isparam(pl, 'key')
%
% VERSION:     $Id: isparam.m,v 1.3 2008/02/14 08:29:09 mauro Exp $
%
% HISTORY:     07-11-2007 Diepholz
%                 Creation
%
%%%
```

isvalid

PURPOSE ^

ISVALID tests if the given plist has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given plist has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given plist has all the correct fields of the
              correct type.

 CALL:        isvalid(plist);

              Returns 1 is plist is a valid PLIST-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(plist, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(plist, 'Version')

              The following call returns the category:
              >> ver = isvalid(plist, 'Category')

 VERSION:     $Id: isvalid.m,v 1.1 2008/02/14 13:48:34 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- plist PLIST Plist class object constructor.

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given plist has all the correct fields of the correct type.
%
%%%
```

```
end

objs = [];
for j=1:nargin
  if isa(varargin{j}, 'plist')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)                      && ...
       (isa (obj.params, 'param')     || ...
        isempty(obj.params))                   && ...
     ischar    (obj.version)                   && ...
     isa       (obj.created, 'time')           && ...
       (isa (obj.plist, 'plist')      || ...
        isempty(obj.plist))

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for param objects.

SYNOPSIS ^

function result = ne(p1, p2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for param objects.

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

 DESCRIPTION: NE overloads the ~= operator for param objects.

 CALL:        result = ne(p1,p2)
              result = ne(p1,p2, 'property1', 'property2')
              result = ne(p1,p2, 'class/property', 'class/property')
              result = ne(p1,p2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      p1,p2 - input param objects

 OUTPUTS:     If the two param objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.5 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a parameter list into a string.
- eq EQ overloads the == operator for plist objects.
- plist PLIST Plist class object constructor.

This function is called by:

SOURCE CODE ^

```
function result = ne(p1, p2, varargin)
% NE overloads the ~= operator for param objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: NE overloads the ~= operator for param objects.
%
% CALL:        result = ne(p1,p2)
%              result = ne(p1,p2, 'property1', 'property2')
%              result = ne(p1,p2, 'class/property', 'class/property')
%              result = ne(p1,p2,  plist('Exceptions', 'exception-list'))
%
% PLIST:       key:   'Exceptions'
%              value:  String or cell-array with exceptions.
%
% INPUTS:      p1,p2 - input param objects
%
% OUTPUTS:     If the two param objects are considered equal, result == 0,
%              otherwise, result == 1.
%
% VERSION:     $Id: ne.m,v 1.5 2008/02/18 15:12:58 ingo Exp $
%
% HISTORY:     29-08-2007 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: ne.m,v 1.5 2008/02/18 15:12:58 ingo Exp $';
CATEGORY = 'Relational Operator';

% Check if this is a call for parameters
if nargin == 2
  if isa(p1, 'plist') && ischar(p2)
    in = char(p2);
    if strcmp(in, 'Params')
      result = plist;
      return
    elseif strcmp(in, 'Version')
      result = VERSION;
      return
    elseif strcmp(in, 'Category')
      result = CATEGORY;
      return
    end
  end
end
```

```

if eq(p1, p2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

nparams

PURPOSE ^

NPARAMS returns the number of param objects in the list.

SYNOPSIS ^

function n = nparams(pl, varargin)

DESCRIPTION ^

```
 NPARAMS returns the number of param objects in the list.

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

 DESCRIPTION: NPARAMS returns the number of param objects in the list.

 CALL:        n = nparams(plist)

 VERSION:     $Id: nparams.m,v 1.5 2008/02/22 22:28:09 hewitson Exp $

 HISTORY:     30-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a parameter list into a string.
- plist PLIST Plist class object constructor.

This function is called by:

- append APPEND append a parameter to the parameter list.
- combine COMBINE multiple parameter lists (plist objects) into a single plist.
- display DISPLAY display plist object.
- remove REMOVE remove a parameter from the parameter list.

SOURCE CODE ^

```
function n = nparams(pl, varargin)
% NPARAMS returns the number of param objects in the list.
%
%%%
```

pset

PURPOSE ^

PSET set parameter list properties.

SYNOPSIS ^

function pl = pset(pl, varargin)

DESCRIPTION ^

```
 PSET set parameter list properties.

%%%
```

```
%%%
```

```

    else
      error('### There is only one key[%s] without a value left.',char(propArgin{1}));
    end

    % check to see if this is one of the parameters we can set, otherwise
    % add it
    found = 0;
    for j=1:length(pl.params)
      if strcmpi(pl.params(j).key, prop)
        pl.params(j).key = prop;
        pl.params(j).val = val;
        found = 1;
      end
    end
    % add this parameter if necessary
    if ~found
      pl = append(pl, param(prop, val));
    end

  end

else
  if ischar(varargin{2})
    error('### please use: %s = set(%s, ''%s'', ''%s'');', ...
                          inputname(1), ...
                          inputname(1), ...
                          varargin{1},  ...
                          varargin{2});
  elseif isnumeric(varargin{2})
    error('### please use: %s = set(%s, ''%s'', %d);', ...
                          inputname(1), ...
                          inputname(1), ...
                          varargin{1},  ...
                          varargin{2});
  else
    error('### please use: %s = set(%s, key, ''value'');', ...
                          inputname(1), ...
                          inputname(1));
  end
end

% END
```

pzmresp

PURPOSE ^

PZMRESP shadow function for the pzmodel/resp.

SYNOPSIS ^

function a = pzmresp(pl, varargin)

DESCRIPTION ^

```
 PZMRESP shadow function for the pzmodel/resp.

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

 DESCRIPTION: PZMRESP shadow function for the pzmodel/resp.
              A parameter called 'pzmodel' is extracted and passed with the
              remaining parameter list to pzmodel/resp.

 CALL:        resp_pzm = pzmresp(pl)

 VERSION:     $Id: pzmresp.m,v 1.4 2008/02/14 08:29:09 mauro Exp $

 HISTORY:     19-04-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a parameter list into a string.
- find FIND overloads find routine for a parameter list.
- plist PLIST Plist class object constructor.
- resp RESP shadows miir/iirResp.

This function is called by:

SOURCE CODE ^

```
function a = pzmresp(pl, varargin)
% PZMRESP shadow function for the pzmodel/resp.
```

```
%
%%%
```

resp

PURPOSE ^

RESP shadows miir/iirResp.

SYNOPSIS ^

function [varargout] = resp(varargin)

DESCRIPTION ^

```
 RESP shadows miir/iirResp.

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

 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);

 VERSION:     $Id: resp.m,v 1.9 2008/02/22 15:27:46 ingo Exp $

 HISTORY:     12-04-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a parameter list into a string.
- find FIND overloads find routine for a parameter list.
- plist PLIST Plist class object constructor.
- resp RESP shadows miir/iirResp.

This function is called by:

- pzmresp PZMRESP shadow function for the pzmodel/resp.
- resp RESP shadows miir/iirResp.

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function [varargout] = resp(varargin)
% RESP shadows miir/iirResp.
%
%%%
```

```
end


pl = varargin{1};
if ~isa(pl, 'plist')
  error('### iirResp needs an input plist.')
end

filt = find(pl, 'filter');
pzm = find(pl, 'pzmodel');
if ~isempty(filt)
  if isa(filt, 'miir') || isa(filt, 'mfir')
    b = resp(filt, pl);
  else
    error('### Unknown filter object.');
  end
elseif ~isempty(pzm)
  if isa(pzm, 'pzmodel')
    b = resp(pzm, pl);
  else
    error('### Unknown pzmodel object.');
  end
else
  error('### input parameter list must contain a ''filter'' or a ''pzmodel'' parameter.');
end

% Set output plist
% b.plist = pl;

% Outputs
if nargout == 0
  plot(b)
end

if nargout == 1
  varargout{1} = b;
end
if nargout > 1
  error('incorrect output arguments');
end


%%%
```

```
%%%
```

save

PURPOSE ^

SAVE a plist object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE a plist object to file.

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

 DESCRIPTION: SAVE overloads the save method for plist objects.

 CALL:        Save a plist object as a .mat file.
                >> save(p, 'blah.mat')
              Save a plist object as an XML file.
                >> save(p, 'blah.xml')
              Save a plist object to file specified in plist.
                >> save(p, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.3 2008/02/14 08:29:09 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(plist, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(plist,'Version')

 The following call returns a string that contains the routine category:

 >> category = save(plist,'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- append APPEND append a parameter to the parameter list.
- char CHAR convert a parameter list into a string.
- plist PLIST Plist class object constructor.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE a plist object to file.
%
%%%
```

```
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

set

PURPOSE ^

SET set parameter list properties.

SYNOPSIS ^

function pl = set(pl, varargin)

DESCRIPTION ^

```
 SET set parameter list properties.

%%%
```

```
% SET set parameter list properties.
%
%%%
```

string

PURPOSE ^

STRING converts a plist object to a command string which will recreate the plist object.

SYNOPSIS ^

function pstr = string(pl, 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)

 VERSION:     $Id: string.m,v 1.12 2008/02/23 19:49:11 hewitson 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:

- string STRING converts a plist object to a command string which will recreate the plist object.

SOURCE CODE ^

```
function pstr = string(pl, varargin)
```

```
% STRING converts a plist object to a command string which will recreate the plist object.
%
%%%
```

```
      elseif isa(p.val, 'history')
        pstr = [pstr '''' p.key ''', ' string(p.val) ', '];
      elseif isa(p.val, 'miir')
        pstr = [pstr '''' p.key ''', ' string(p.val) ', '];
      elseif isa(p.val, 'mfir')
        pstr = [pstr '''' p.key ''', ' string(p.val) ', '];
      elseif isa(p.val, 'specwin')
        pstr = [pstr '''' p.key ''', ' string(p.val) ', '];
      elseif isa(p.val, 'pole') || isa(p.val, 'zero') || isa(p.val, 'pzmodel')
        pstr = [pstr '''' p.key ''', ' string(p.val) ', '];
      elseif isa(p.val, 'time')
        pstr = [pstr '''' p.key ''', ' string(p.val) ', '];
      elseif isa(p.val, 'plist')
        pstr = [pstr '''' p.key ''', ' string(p.val) ', '];
      elseif isa(p.val, 'timeformat')
        pstr = [pstr '''' p.key ''', ' string(p.val) ', '];
      elseif isa(p.val, 'timespan')
        pstr = [pstr '''' p.key ''', ' string(p.val) ', '];
      elseif isa(p.val, 'param')
        pstr = [pstr '''' p.key ''', ' string(p.val) ', '];
      elseif isa(p.val, 'provenance')
        pstr = [pstr '''' p.key ''', ' string(p.val) ', '];
      elseif iscell(p.val)
        val_str = ['{ '];
        for ii = 1:numel(p.val)
          cell_val = p.val{ii};
          if ischar(cell_val)
            val_str = [val_str '''' p.val{ii} ''', ' ];
          elseif isnumeric(cell_val)
            val_str = [val_str ltpda_mat2str(p.val{ii}) ', ' ];
          elseif isobject(cell_val)
            val_str = [val_str string(p.val{ii}) ', ' ];
          else
            error('### Unknown object in cell array');
          end
        end
        val_str = val_str(1:end-2);
        val_str = [val_str ' }'];
        pstr = [pstr '''' p.key ''', ' val_str ', '];
      elseif isa(p.val, 'cdata') || isa(p.val, 'fsdata') || ...
          isa(p.val, 'tsdata')|| isa(p.val, 'xydata') || ...
          isa(p.val, 'xyzdata')
        pstr = [pstr '''' p.key ''', ' class(p.val) '(), '];
      else
        error('!!! Can not handle the class [%s].', class(p.val));
      end
    end

    % close bracket
    if ~isempty(pstr)
      pstr = pstr(1:end-2);
```

```
    end
    pstr = [pstr ')'];
  else
    pstr = 'plist()';
  end
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsasgn

PURPOSE ^

SUBSASGN define index assignment for parameter list properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for parameter list properties.

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

 DESCRIPTION: SUBSASGN define index assignment for parameter list
              properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.9 2008/02/25 18:06:31 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a parameter list into a string.
- plist PLIST Plist class object constructor.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for parameter list properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
% DESCRIPTION: SUBSASGN define index assignment for parameter list
%              properties.
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.9 2008/02/25 18:06:31 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

subsref

PURPOSE ^

SUBSREF Define field name indexing for parameter list objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for parameter list objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for parameter list objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.15 2008/02/22 22:26:40 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a parameter list into a string.
- plist PLIST Plist class object constructor.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for parameter list objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for parameter list objects.
```

```
%
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.15 2008/02/22 22:26:40 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

< Master index Index for classes/@param >

Index for classes/@param

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: param %%%%%%%%%%%%%%%%%%%%%%
cat	CAT concatonate params into a vector.
char	CHAR convert a param object into a string.
display	DISPLAY display a parameter
eq	EQ overloads the == operator for param objects.
get	GET get parameter properties.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given param has all the correct fields of the correct type.
mux	MUX concatenate params into a vector.
ne	NE overloads the ~= operator for param objects.
param	PARAM Parameter object class constructor.
save	SAVE a param object to file.
set	SET set a parameter property.
string	STRING writes a command string that can be used to recreate the input param object.
subsasgn	SUBSASGN define index assignment for parameter properties.
subsref	SUBSREF Define field name indexing for parameter objects.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

char

PURPOSE ^

CHAR convert a param object into a string.

SYNOPSIS ^

function pstr = char(p, varargin)

DESCRIPTION ^

```
 CHAR convert a param object into a string.

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

 DESCRIPTION: CHAR convert a param object into a string.

 CALL:        string = char(p)

 VERSION:     $Id: char.m,v 1.6 2008/02/13 17:33:38 mauro Exp $

 HISTORY:     04-02-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- get GET get parameter properties.

This function is called by:

- cat CAT concatonate params into a vector.
- char CHAR convert a param object into a string.
- display DISPLAY display a parameter
- mux MUX concatenate params into a vector.
- ne NE overloads the ~= operator for param objects.
- param PARAM Parameter object class constructor.
- save SAVE a param object to file.
- string STRING writes a command string that can be used to recreate the input param object.

- subsasgn SUBSASGN define index assignment for parameter properties.
- subsref SUBSREF Define field name indexing for parameter objects.

SOURCE CODE ^

```
function pstr = char(p, varargin)
% CHAR convert a param object into a string.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: CHAR convert a param object into a string.
%
% CALL:        string = char(p)
%
% VERSION:     $Id: char.m,v 1.6 2008/02/13 17:33:38 mauro Exp $
%
% HISTORY:     04-02-2007 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION = '$Id: char.m,v 1.6 2008/02/13 17:33:38 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(p, 'param') && ischar(varargin{1})
    in = char(varargin{1});
    if strcmp(in, 'Params')
      pstr = plist;
      return
    elseif strcmp(in, 'Version')
      pstr = VERSION;
      return
    elseif strcmp(in, 'Category')
      pstr = CATEGORY;
      return
    end
  end
end

if isnumeric(get(p, 'val'))
  pstr = [get(p, 'key') ' : ' ltpda_mat2str(get(p, 'val'))];
else
  pstr = [get(p, 'key') ' : ' char(get(p, 'val'))];
end



% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

save

PURPOSE ^

SAVE a param object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE a param object to file.

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

 DESCRIPTION: SAVE overloads the save method for param objects.

 CALL:        Save a param object as a .mat file.
                >> save(p, 'blah.mat')
              Save a param object as an XML file.
                >> save(p, 'blah.xml')
              Save a param object to file specified in plist.
                >> save(p, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.3 2008/02/13 17:33:38 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(param, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(param, 'Version')

 The following call returns a string that contains the routine category:

 >> category = save(param, 'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- param PARAM Parameter object class constructor.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE a param object to file.
%
%%%
```

```
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsasgn

PURPOSE ^

SUBSASGN define index assignment for parameter properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for parameter properties.

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

 DESCRIPTION: SUBSASGN define index assignment for parameter properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.7 2008/02/25 18:06:31 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- param PARAM Parameter object class constructor.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for parameter properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSASGN define index assignment for parameter properties.
```

```
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.7 2008/02/25 18:06:31 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

< 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.
display	DISPLAY overloads display functionality for specwin objects.
eq	EQ overloads the == operator for specfin objects.
get	GET get specwin object properties.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given specwin has all the correct fields of the correct type.
ne	NE overloads the ~= operator for specwin objects.
plot	PLOT plots a specwin object.
save	SAVE a specwin object to file.
set	SET sets a specwin property.
specwin	SPECWIN spectral window object class constructor.
string	STRING writes a command string that can be used to recreate the input window object.
subsasgn	SUBSASGN define index assignment for specwin properties.
subsref	SUBSREF Define field name indexing for specwin objects.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

char

PURPOSE ^

CHAR convert a specwin object into a string.

SYNOPSIS ^

function pstr = char(w, varargin)

DESCRIPTION ^

```
 CHAR convert a specwin object into a string.

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

 DESCRIPTION: CHAR convert a specwin object into a string.

 CALL:        string = char(sw);

 VERSION:     $Id: char.m,v 1.4 2008/02/13 13:46:07 mauro Exp $

 HISTORY:     04-02-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a specwin object into a string.

This function is called by:

- char CHAR convert a specwin object into a string.
- display DISPLAY overloads display functionality for specwin objects.
- ne NE overloads the ~= operator for specwin objects.
- plot PLOT plots a specwin object.
- save SAVE a specwin object to file.
- specwin SPECWIN spectral window object class constructor.
- string STRING writes a command string that can be used to recreate the input window object.
- subsasgn SUBSASGN define index assignment for specwin properties.
- subsref SUBSREF Define field name indexing for specwin objects.

SOURCE CODE ^

```
function pstr = char(w, varargin)
% CHAR convert a specwin object into a string.
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

set

PURPOSE ^

SET sets a specwin property.

SYNOPSIS ^

function w = set(w, varargin)

DESCRIPTION ^

```
 SET sets a specwin property.

%%%
```

- plot PLOT plots a specwin object.
- specwin SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
function w = set(w, varargin)
% SET sets a specwin property.
%
%%%
```

```
if (nargout == 0)
  display(w);
  error('\n### No output variable!\n### Please use: %s = set(%s, ''key'', ''value'');', inputname(1), inputname(1));
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsasgn

PURPOSE ^

SUBSASGN define index assignment for specwin properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for specwin properties.

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

 DESCRIPTION: SUBSASGN define index assignment for specwin properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.5 2008/02/25 18:06:32 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a specwin object into a string.
- specwin SPECWIN spectral window object class constructor.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for specwin properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSASGN define index assignment for specwin properties.
```

```
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.5 2008/02/25 18:06:32 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

< Master index Index for classes/@pole >

Index for classes/@pole

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: pole %%%%%%%%%%%%%%%%%%%%%%
cat	CAT concatonate poles into a vector.
char	CHAR convert a pole object into a string.
cp2iir	CP2IIR Return a,b IIR filter coefficients for a complex pole designed using the bilinear transform.
display	DISPLAY overloads display functionality for pole objects.
eq	EQ overloads the == operator for pole objects.
get	GET get pole properties.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given pole has all the correct fields of the correct type.
ne	NE overloads the ~= operator for pole objects.
pole	POLE construct a pole object.
resp	RESP returns the complex response of the pole object.
rp2iir	RP2IIR Return a,b coefficients for a real pole designed using the bilinear transform.
save	SAVE a pole object to file.
set	SET set a pole property.
string	STRING writes a command string that can be used to recreate the input pole object.
subsasgn	SUBSASGN define index assignment for pole properties.
subsref	SUBSREF Define field name indexing for pole objects.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

char

PURPOSE ^

CHAR convert a pole object into a string.

SYNOPSIS ^

function pstr = char(p, varargin)

DESCRIPTION ^

```
 CHAR convert a pole object into a string.

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

 DESCRIPTION: CHAR convert a pole object into a string.

 CALL:        string = char(pole);

 VERSION:     $Id: char.m,v 1.4 2008/02/12 21:38:23 hewitson Exp $

 HISTORY:     02-04-2007 M Hewitson
                  Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pole object into a string.

This function is called by:

- cat CAT concatonate poles into a vector.
- char CHAR convert a pole object into a string.
- display DISPLAY overloads display functionality for pole objects.
- ne NE overloads the ~= operator for pole objects.
- pole POLE construct a pole object.
- resp RESP returns the complex response of the pole object.
- save SAVE a pole object to file.
- subsasgn SUBSASGN define index assignment for pole properties.
- subsref SUBSREF Define field name indexing for pole objects.

SOURCE CODE ^

```
function pstr = char(p, varargin)
% CHAR convert a pole object into a string.
%
%%%
```

subsasgn

PURPOSE ^

SUBSASGN define index assignment for pole properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for pole properties.

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

 DESCRIPTION: SUBSASGN define index assignment for pole properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.5 2008/02/25 18:06:31 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pole object into a string.
- pole POLE construct a pole object.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for pole properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSASGN define index assignment for pole properties.
```

```
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.5 2008/02/25 18:06:31 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

< Master index Index for classes/@zero >

Index for classes/@zero

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: zero %%%%%%%%%%%%%%%%%%%%%%
cat	CAT concatonate zeros into a vector.
char	CHAR convert a zero object into a string.
cz2iir	CZ2IIR return a,b IIR filter coefficients for a complex zero designed using the bilinear transform.
display	DISPLAY overloads display functionality for zero objects.
eq	EQ overloads the == operator for zero objects.
get	GET get zero properties.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given zero has all the correct fields of the correct type.
ne	NE overloads the ~= operator for zero objects.
resp	RESP returns the complex response of the zero object.
rz2iir	RZ2IIR Return a,b IIR filter coefficients for a real zero designed using the bilinear transform.
save	SAVE a zero object to file.
set	SET set a zero property.
string	STRING writes a command string that can be used to recreate the input zero object.
subsasgn	SUBSASGN define index assignment for zero properties.
subsref	SUBSREF Define field name indexing for zero objects.
zero	ZERO construct a pole object.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

char

PURPOSE ^

CHAR convert a zero object into a string.

SYNOPSIS ^

function pstr = char(p, varargin)

DESCRIPTION ^

```
 CHAR convert a zero object into a string.

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

 DESCRIPTION: CHAR convert a zero object into a string.

 CALL:        string = char(zero);

 VERSION:     $Id: char.m,v 1.4 2008/02/12 19:54:16 hewitson Exp $

 HISTORY:     02-04-2007 Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a zero object into a string.

This function is called by:

- cat CAT concatonate zeros into a vector.
- char CHAR convert a zero object into a string.
- display DISPLAY overloads display functionality for zero objects.
- ne NE overloads the ~= operator for zero objects.
- resp RESP returns the complex response of the zero object.
- save SAVE a zero object to file.
- subsasgn SUBSASGN define index assignment for zero properties.
- subsref SUBSREF Define field name indexing for zero objects.
- zero ZERO construct a pole object.

SOURCE CODE ^

```
function pstr = char(p, varargin)
% CHAR convert a zero object into a string.
%
%%%
```

subsasgn

PURPOSE ^

SUBSASGN define index assignment for zero properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for zero properties.

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

 DESCRIPTION: SUBSASGN define index assignment for zero properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:32 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a zero object into a string.
- zero ZERO construct a pole object.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for zero properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSASGN define index assignment for zero properties.
```

```
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:32 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

< Master index Index for classes/@pzmodel >

Index for classes/@pzmodel

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: pzmodel %%%%%%%%%%%%%%%%%%%%%
char	CHAR convert a pzmodel object into a string.
display	DISPLAY overloads display functionality for pzmodel objects.
eq	EQ overloads the == operator for pzmodel objects.
get	GET get a pole/zero model property.
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.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given pzmodel has all the correct fields of the correct type.
ne	NE overloads the ~= operator for pzmodel objects.
pzmodel	PZMODEL constructor for pzmodel class.
resp	RESP returns the complex response of a pzmodel as an Analysis Object.
save	SAVE a pzmodel object to file.
set	SET sets a pole/zero model property.
string	STRING writes a command string that can be used to recreate the input pzmodel object.
subsasgn	SUBSASGN define index assignment for pzmodel properties.
subsref	SUBSREF Define field name indexing for pole/zero model objects.
tomfir	TOMFIR approximates a pole/zero model with an FIR filter.
tomiir	TOMIIR converts a pzmodel to an IIR filter using a bilinear transform.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

char

PURPOSE ^

CHAR convert a pzmodel object into a string.

SYNOPSIS ^

function pstr = char(p, varargin)

DESCRIPTION ^

```
 CHAR convert a pzmodel object into a string.

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

 DESCRIPTION: CHAR convert a pzmodel object into a string.

 CALL:        string = char(pzmodel)

 VERSION:     $Id: char.m,v 1.4 2008/02/12 20:07:37 hewitson Exp $

 HISTORY:     02-04-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pzmodel object into a string.

This function is called by:

- char CHAR convert a pzmodel object into a string.
- display DISPLAY overloads display functionality for pzmodel objects.
- 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.
- ne NE overloads the ~= operator for pzmodel objects.
- pzmodel PZMODEL constructor for pzmodel class.
- resp RESP returns the complex response of a pzmodel as an Analysis Object.
- save SAVE a pzmodel object to file.
- string STRING writes a command string that can be used to recreate the input pzmodel object.

- subsasgn SUBSASGN define index assignment for pzmodel properties.
- subsref SUBSREF Define field name indexing for pole/zero model objects.
- tomfir TOMFIR approximates a pole/zero model with an FIR filter.
- tomiir TOMIIR converts a pzmodel to an IIR filter using a bilinear transform.

SOURCE CODE ^

```
function pstr = char(p, varargin)
% CHAR convert a pzmodel object into a string.
%
%%%
```

set

PURPOSE ^

SET sets a pole/zero model property.

SYNOPSIS ^

function pzm = set(pzm, varargin)

DESCRIPTION ^

```
 SET sets a pole/zero model property.

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

 DESCRIPTION: SET sets a pole/zero model property.

 CALL:        pzmodel = set(pzmodel, 'name',    str);
              pzmodel = set(pzmodel, 'gain',    g);
              pzmodel = set(pzmodel, 'poles',   p);
              pzmodel = set(pzmodel, 'zeros',   z);
              pzmodel = set(pzmodel, 'created', time-object);
              pzmodel = set(pzmodel, 'version', cvs version string);

 VERSION:     $Id: set.m,v 1.6 2008/02/13 18:33:19 ingo Exp $

 HISTORY:     30-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- display DISPLAY overloads display functionality for pzmodel objects.

This function is called by:

- 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.

SOURCE CODE ^

```
function pzm = set(pzm, varargin)
% SET sets a pole/zero model property.
%
%%%
```

subsasgn

PURPOSE ^

SUBSASGN define index assignment for pzmodel properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for pzmodel properties.

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

 DESCRIPTION: SUBSASGN define index assignment for pzmodel properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:31 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pzmodel object into a string.
- pzmodel PZMODEL constructor for pzmodel class.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for pzmodel properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSASGN define index assignment for pzmodel properties.
```

```
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:31 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

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 operator for Analysis objects.
   <a href="matlab:help ao/acos">ao/acos</a>               -  overloads the acos
operator for Analysis objects. Inverse cosine result in radians.
   <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 contained in the analysis object.
   <a href="matlab:help ao/aosplit">ao/aosplit</a>            -  an analysis object
into the specified segments.
   <a href="matlab:help ao/asin">ao/asin</a>               -  overloads the asin
operator for Analysis objects. Inverse sine result in radians.
   <a href="matlab:help ao/atan">ao/atan</a>               -  overloads the atan
operator for Analysis objects. Inverse tangent result in radians.
   <a href="matlab:help ao/attachm">ao/attachm</a>            -  attach the m file
being executes 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>                -  concatonate AOs into a
vector.
   <a href="matlab:help ao/char">ao/char</a>               -  overloads char()
function for analysis objects.
   <a href="matlab:help ao/complex">ao/complex</a>            -  overloads the
complex operator for Analysis objects.
   <a href="matlab:help ao/conj">ao/conj</a>               -  overloads the conjugate
operator for Analysis objects.
   <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/ctranspose">ao/ctranspose</a>         -  overloads the '
operator for Analysis Objects.
   <a href="matlab:help ao/decimate">ao/decimate</a>           -  AOs containing time-
series data.
   <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/diag">ao/diag</a>              -  overloads the diagonal
operator for Analysis Objects.
   <a href="matlab:help ao/display">ao/display</a>           -  implement terminal
display for analysis object.
   <a href="matlab:help ao/eig">ao/eig</a>               -  overloads the
determinant function for Analysis objects.
   <a href="matlab:help ao/eq">ao/eq</a>                -  overloads the == operator
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/fft">ao/fft</a>               -  overloads the fft
operator for Analysis objects.
   <a href="matlab:help ao/filter">ao/filter</a>            -  overides the filter
function for analysis objects.
   <a href="matlab:help ao/filtfilt">ao/filtfilt</a>          -  overides 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/ge">ao/ge</a>                -  overloads >= operator for
analysis objects. Compare the y-axis values.
   <a href="matlab:help ao/get">ao/get</a>               -  get ao properties.
   <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/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 specified by the input parameter list.
   <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>             -  plots an array of
analysis objects.
   <a href="matlab:help ao/isfield">ao/isfield</a>           -  tests if the given
field is one of the object properties.
   <a href="matlab:help ao/join">ao/join</a>              -  multiple AOs into a
single AO.
   <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/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/lt">ao/lt</a>                -  overloads < operator for
analysis objects. Compare the y-axis values.
   <a href="matlab:help ao/md5">ao/md5</a>               -  computes an MD5 checksum
from an analysis objects.
   <a href="matlab:help ao/mean">ao/mean</a>              -  overloads the mean
operator for Analysis objects. Compute the mean value.
   <a href="matlab:help ao/median">ao/median</a>            -  overloads the median
operator for Analysis objects. Compute the median value.
   <a href="matlab:help ao/minus">ao/minus</a>             -  overloads - operator
for analysis objects.
   <a href="matlab:help ao/mpower">ao/mpower</a>            -  implements power
operator for analysis objects.
   <a href="matlab:help ao/mrdivide">ao/mrdivide</a>          -  overloads /
operator for analysis objects.
   <a href="matlab:help ao/mtimes">ao/mtimes</a>            -  overloads * operator
for analysis objects.
   <a href="matlab:help ao/mux">ao/mux</a>               -  concatonate AOs into a
vector.
   <a href="matlab:help ao/ne">ao/ne</a>                -  overloads the ~= operator
for analysis objects.
   <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/plot_old">ao/plot_old</a>          -  plots an array of
analysis objects.
   <a href="matlab:help ao/plus">ao/plus</a>              -  implements addition
operator for two 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/rdivide">ao/rdivide</a>           -  overloads ./
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/save">ao/save</a>              -  overloads save operator
for analysis objects.
   <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/set">ao/set</a>               -  set an analysis object
property.
   <a href="matlab:help ao/setnh">ao/setnh</a>             -  set an analysis object
property without recording the history.
   <a href="matlab:help ao/simple_plot">ao/simple_plot</a>       -  a simple plot of
analysis objects.
   <a href="matlab:help ao/sin">ao/sin</a>               -  overloads the sin
operator for Analysis objects. Sine of argument in radians.
```

```
   <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>              -  overloads the sqrt
operator for Analysis objects. Square root.
   <a href="matlab:help ao/stairs">ao/stairs</a>            -  overloads the stairs
function for Analysis Objects.
   <a href="matlab:help ao/std">ao/std</a>               -  overloads the std
operator for Analysis objects. Compute the standard deviation.
   <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/submit">ao/submit</a>            -  AO(s) to the MySQL
server.
   <a href="matlab:help ao/submit_fastinsert">ao/submit_fastinsert</a> -  AO(s) to
the MySQL server.
   <a href="matlab:help ao/subsasgn">ao/subsasgn</a>          -  define index
assignment for analysis object properties.
   <a href="matlab:help ao/subsref">ao/subsref</a>           -  Define field name
indexing for analysis objects.
   <a href="matlab:help ao/sum">ao/sum</a>               -  overloads the sum
operator for Analysis objects. Compute the sum value.
   <a href="matlab:help ao/svd">ao/svd</a>               -  overloads the
determinant function for Analysis objects.
   <a href="matlab:help ao/tag">ao/tag</a>               -  tag each input AO with a
proper ID number.
   <a href="matlab:help ao/tan">ao/tan</a>               -  overloads the tan
operator for Analysis objects. Tangent of argument in radians.
   <a href="matlab:help ao/testAO">ao/testAO</a>            -  checks that an
analysis object has the correct class fields.
   <a href="matlab:help ao/times">ao/times</a>             -  overloads .* operator
for analysis objects.
   <a href="matlab:help ao/transpose">ao/transpose</a>         -  overloads the .'
operator for Analysis Objects.
   <a href="matlab:help ao/var">ao/var</a>               -  overloads the var
operator for Analysis objects. Compute the variance.
   <a href="matlab:help ao/xml">ao/xml</a>               -  convert an analysis
object to an DOM XML object.
   <a href="matlab:help ao/xmladd">ao/xmladd</a>            -  add an analysis
object to the given node of the input XML DOM
   <a href="matlab:help ao/xmlparse">ao/xmlparse</a>          -  creates a ao from a
ao xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ao2m

PURPOSE ^

AO2M converts an analysis object to an '.m' file based on the history contained in the analysis object.

SYNOPSIS ^

function varargout = ao2m(varargin)

DESCRIPTION ^

```
 AO2M converts an analysis object to an '.m' file based on the history contained in
the analysis object.

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

 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 1.14 2008/02/23 18:06:31 hewitson Exp $

 HISTORY: 08-02-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.
- find FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

SOURCE CODE ^

```
function varargout = ao2m(varargin)
% AO2M converts an analysis object to an '.m' file based on the history contained in the analysis object.
%
%%%
```

```
%% Get inputs and proceed
a        = varargin{1};
% get filename from plist
if isa(varargin{2}, 'plist')
  filename = find(varargin{2}, 'filename');
else % or directly
  filename = varargin{2};
end
if isempty(filename)
  error('### You must specify a filename. Either directly, or in a plist');
end

if length(a) > 1
  error('### I can only deal with one AO at a time.');
end


% clear this m-file from memory
clear(filename);

% convert the history to a set of commands
cmds = hist2m(a.hist);

% Now write output file
[path,name,ext,vers] = fileparts(filename);
if isempty(ext)
  filename = [filename '.m'];
end
if ~strcmp(ext, '.m')
  error('### I can only write .m files.');
end

d = a.data;
fd = fopen(filename, 'w+');
fprintf(fd, 'function a_out = %s\n\n', name);
fprintf(fd, '%% %s \n', upper(filename));
fprintf(fd, '%% \n');
fprintf(fd, '%% \n');
fprintf(fd, '%% written by %s / %s\n', ALGONAME, VERSION);
fprintf(fd, '%% \n');
fprintf(fd, '%% based on analysis object:\n');
fprintf(fd, '%%            name: %s / %s\n', a.name, d.name);
fprintf(fd, '%%      provenance: %s\n', char(a.provenance));
fprintf(fd, '%%     description: %s\n', char(a.description));
fprintf(fd, '%% original m-file: %s\n', a.mfilename);
fprintf(fd, '%% \n');
fprintf(fd, '%% \n');
fprintf(fd, ' \n');
fprintf(fd, ' \n');
for c=length(cmds):-1:1
  if ~isempty(cmds{c})
    fprintf(fd, '%s\n', cmds{c});
```

```
  end
end
fprintf(fd, ' \n');
fprintf(fd, ' \n');
fprintf(fd, '%% END\n');

fclose(fd);

cd ('.');


```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

attachm

PURPOSE ^

ATTACHM attach the m file being executes to the analysis object.

SYNOPSIS ^

function a = attachm(a, mfile)

DESCRIPTION ^

```
 ATTACHM attach the m file being executes to the analysis object.

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

 DESCRIPTION: ATTACHM attach the m file being executes to the analysis object.

 CALL:        ao = attachm(ao, 'foo.m');

 VERSION:     $Id: attachm.m,v 1.6 2008/02/23 20:49:01 hewitson Exp $

 HISTORY:     31-03-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.

This function is called by:

SOURCE CODE ^

```
function a = attachm(a, mfile)
% ATTACHM attach the m file being executes to the analysis object.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: ATTACHM attach the m file being executes to the analysis object.
%
```

```
% CALL:        ao = attachm(ao, 'foo.m');
%
% VERSION:     $Id: attachm.m,v 1.6 2008/02/23 20:49:01 hewitson Exp $
%
% HISTORY:     31-03-07 M Hewitson
%                 Creation
%
%%%
```

attachmdl

PURPOSE ^

ATTACHMDL attach an mdl file to the analysis object.

SYNOPSIS ^

function a = attachmdl(varargin)

DESCRIPTION ^

```
 ATTACHMDL attach an mdl file to the analysis object.

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

 DESCRIPTION: ATTACHMDL attach an mdl file to the analysis object.

 CALL:        a = attachmdl(a)

 VERSION:     $Id: attachmdl.m,v 1.5 2008/02/23 18:07:00 hewitson Exp $

 HISTORY: 31-03-07 M Hewitson
            Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function a = attachmdl(varargin)
% ATTACHMDL attach an mdl file to the analysis object.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: ATTACHMDL attach an mdl file to the analysis object.
%
% CALL:        a = attachmdl(a)
%
```

```
% VERSION:     $Id: attachmdl.m,v 1.5 2008/02/23 18:07:00 hewitson Exp $
%
% HISTORY: 31-03-07 M Hewitson
%             Creation
%
%%%
```

cat

PURPOSE ^

CAT concatonate AOs into a vector.

SYNOPSIS ^

function b = cat(varargin)

DESCRIPTION ^

```
 CAT concatonate AOs into a vector.

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

 DESCRIPTION: CAT concatenate AOs into a vector.

 CALL:        a = cat(a1, a2)

 VERSION:     $Id: cat.m,v 1.7 2008/02/12 10:11:13 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = cat(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = cat(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = cat(ao,'Category')

 HISTORY: 27-03-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.

This function is called by:

SUBFUNCTIONS ^

- function pl_default = getDefaultPL()

SOURCE CODE ^

```
function b = cat(varargin)
% CAT concatonate AOs into a vector.
%
%%%
```

```
    elseif strcmp(in, 'Category')
      b = CATEGORY;
      return
    end
  end
end

%% capture input variable names
invars = {};
for j=1:nargin
  iname = inputname(j);
  if isempty(iname) && isnumeric(varargin{j})
    iname = num2str(varargin{j});
  elseif isempty(iname) && ischar(varargin{j})
    iname = varargin{j};
  end
  invars = [invars cellstr(iname)];
end

b = [];
for j=1:nargin
  if isa(varargin{j}, 'ao')
    b = [b varargin{j}];
  end
end

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist();


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

demux

PURPOSE ^

DEMUX splits the input vector of AOs into a number of output AOs.

SYNOPSIS ^

function varargout = demux(as, params)

DESCRIPTION ^

```
 DEMUX splits the input vector of AOs into a number of output AOs.

%%%
```

This function is called by:

SUBFUNCTIONS ^

- function pl_default = getDefaultPL()

SOURCE CODE ^

```
function varargout = demux(as, params)
% DEMUX splits the input vector of AOs into a number of output AOs.
%
%%%
```

```
      varargout{1} = CATEGORY;
      return
    end
  end
end

%% get number of input AOs
n  = numel(as);

if nargout > n
  error('### too many output arguments');
elseif nargout > n
  error('### not enough output arguments');
end

for j=1:nargout
  % map to output AO
  varargout{j} = as(j);
end

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist();

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for analysis objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for analysis objects.

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

 DESCRIPTION: EQ overloads the == operator for analysis objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input analysis objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two analysis objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.12 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

export

PURPOSE ^

EXPORT export an analysis object to a text file.

SYNOPSIS ^

function varargout = export(a, fname)

DESCRIPTION ^

```
 EXPORT export an analysis object to a text file.

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

 DESCRIPTION: EXPORT export an analysis object to a text file.

 CALL:        export(a, 'blah.txt');

 VERSION:     $Id: export.m,v 1.7 2008/01/22 20:59:42 hewitson Exp $

 HISTORY: 30-03-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.
- imag IMAG overloads the imaginary operator for Analysis objects.
- real REAL overloads the real operator for Analysis objects.
- save SAVE overloads save operator for analysis objects.

This function is called by:

SOURCE CODE ^

```
function varargout = export(a, fname)
% EXPORT export an analysis object to a text file.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%
% DESCRIPTION: EXPORT export an analysis object to a text file.
%
% CALL:        export(a, 'blah.txt');
%
% VERSION:     $Id: export.m,v 1.7 2008/01/22 20:59:42 hewitson Exp $
%
% HISTORY: 30-03-07 M Hewitson
%             Creation
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

extractm

PURPOSE ^

EXTRACTM extracts an m-file from an analysis object and saves it to

SYNOPSIS ^

function a = extractm(varargin)

DESCRIPTION ^

```
 EXTRACTM extracts an m-file from an analysis object and saves it to
 disk.

%%%
```

```
%               disk.
%
% CALL:         extractm(a)
%
% VERSION:      $Id: extractm.m,v 1.1 2008/02/23 20:50:40 hewitson Exp $
%
% HISTORY: 23-02-08 M Hewitson
%             Creation
%
%%%
```

extractmdl

PURPOSE ^

EXTRACTMDL extracts an mdl file from an analysis object and saves it to

SYNOPSIS ^

function a = 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)

 VERSION:     $Id: extractmdl.m,v 1.1 2008/02/23 18:08:06 hewitson Exp $

 HISTORY: 31-03-07 M Hewitson
            Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function a = extractmdl(varargin)
% 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)
%
% VERSION:      $Id: extractmdl.m,v 1.1 2008/02/23 18:08:06 hewitson Exp $
%
% HISTORY: 31-03-07 M Hewitson
%             Creation
%
%%%
```

ge

PURPOSE ^

GE overloads >= operator for analysis objects. Compare the y-axis values.

SYNOPSIS ^

function a = ge(b, c, 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;

 POSSIBLE VALUES: b: Analysis object
                  c: Analysis object or a scalar.

 VERSION: $Id: ge.m,v 1.7 2008/02/25 18:04:22 ingo Exp $

 HISTORY: 06-02-2007 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.

This function is called by:

SOURCE CODE ^

```
function a = ge(b, c, varargin)
% GE overloads >= operator for analysis objects. Compare the y-axis values.
%
```

```
%%%
```

```
a = x>=y;

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get ao properties.

SYNOPSIS ^

function val = get(aos, propName)

DESCRIPTION ^

```
 GET get ao properties.

%%%
```

```
 HISTORY: 30-01-07 M Hewitson
             Creation

%%%
```

```
%       version     = get(ao, 'version');
%       created     = get(ao, 'created');
%       fs          = get(ao, 'fs');         --> ao.data.fs;
%       t0          = get(ao, 't0');         --> ao.data.t0;
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = get(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = get(ao, 'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = get(ao, 'Category')
%
% VERSION: $Id: get.m,v 1.15 2008/02/13 07:58:13 mauro Exp $
%
% HISTORY: 30-01-07 M Hewitson
%             Creation
%
%%%
```

```
%%% Special case: If possible return the 't0' property of the data object
elseif strcmp(propName, 't0')
  if ismember('t0', data_fields)
    val = aos.data.t0;
    return
  else
    error('### ''t0'' is not a field of the data-property. The data-property is from the class [%s]', class(aos.data));
  end
end

val = generic_get(aos, propName, DEFAULT_PLIST, VERSION, CATEGORY);
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

gt

PURPOSE ^

GT overloads > operator for analysis objects. Compare the y-axis values.

SYNOPSIS ^

function a = gt(b, c, varargin)

DESCRIPTION ^

```
 GT overloads > operator for analysis objects. Compare the y-axis values.

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

 DESCRIPTION: GT overloads > operator for analysis objects.
              Compare the y-axis values.

 CALL: a = b>=;

 POSSIBLE VALUES: b: Analysis object
                  c: Analysis object or a scalar.

 VERSION: $Id: gt.m,v 1.6 2008/02/25 18:04:22 ingo Exp $

 HISTORY: 06-02-2007 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.

This function is called by:

SOURCE CODE ^

```
function a = gt(b, c, varargin)
% GT overloads > operator for analysis objects. Compare the y-axis values.
%
```

```
%%%
```

```

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

index

PURPOSE ^

INDEX index into an AO array or matrix. This properly captures the history.

SYNOPSIS ^

function b = 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)

 VERSION:     $Id: index.m,v 1.8 2008/02/12 19:29:34 hewitson Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = index(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = index(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = index(ao,'Category')

 HISTORY: 02-05-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.
- find FIND particular samples that satisfy the input query and return a new AO.
- get GET get ao properties.
- set SET set an analysis object property.
- setnh SET set an analysis object property without recording the history.

This function is called by:

SUBFUNCTIONS ^

- function pl_default = getDefaultPL()

SOURCE CODE ^

```
function b = index(varargin)
% 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)
%
% VERSION:     $Id: index.m,v 1.8 2008/02/12 19:29:34 hewitson Exp $
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = index(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = index(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = index(ao,'Category')
%
% HISTORY: 02-05-07 M Hewitson
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ALGONAME = mfilename;
VERSION  = '$Id: index.m,v 1.8 2008/02/12 19:29:34 hewitson Exp $';
```

```
CATEGORY = 'Operator';

%% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      b = getDefaultPL();
      return
    elseif strcmp(in, 'Version')
      b = VERSION;
      return
    elseif strcmp(in, 'Category')
      b = CATEGORY;
      return
    end
  end
end

%% capture input variable names
invars = {};
for j=1:nargin
  invars = [invars cellstr(inputname(j))];
end

as = varargin{1};
if ~isa(as, 'ao')
  error('### an array or matrix of AO objects is expected as first input argument.');
end

% Get indices
pl = [];
if nargin == 2
  if isa(varargin{2}, 'plist')
    pl = varargin{2};
    i  = find(pl, 'i');
    j  = find(pl, 'j');
    % Index the AO array
    if isempty(j)
      % Index the AO array
      asi = as(i);
    else
      % Index the AO array
      asi = as(i,j);
    end
  else
    i = varargin{2};
    j = 0;
    % Index the AO array
    asi = as(i);
  end
```

```
elseif nargin == 3
  i = varargin{2};
  j = varargin{3};
  if j == 0
    % Index the AO array
    asi = as(i);
  else
    % Index the AO array
    asi = as(i,j);
  end
else
  error('### incorrect inputs.');
end


si = size(asi);
na = si(1)*si(2);
if na ~= 1
  error('### I only support a single index currently.');
end

b = [];

% Now build output AO
if isempty(pl)
  pl = plist([param('i', i) param('j', j)]);
end
h = history(ALGONAME, VERSION, pl, get(asi,'hist'));
h = set(h, 'invars', invars);

% make output analysis object
aout = setnh(asi, 'hist', h);

b = [b aout];

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist();
  pl_default = append(pl_default, 'i', []);
  pl_default = append(pl_default, 'j', []);


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

iplot

PURPOSE ^

IPLOT provides an intelligent plotting tool for LTPDA.

SYNOPSIS ^

function varargout = iplot(varargin)

DESCRIPTION ^

```
 IPLOT provides an intelligent plotting tool for LTPDA.

%%%
```

```
        lines.

            'LineColors' - a cell array of color definitions for each line.

            'LineStyles' - a cell array of line styles.

            'Markers'    - a cell array of markers.

            'LineWidths' - a cell array of line widths. If the length of the
                           cell array is shorter than the number of lines to
                           plot, the remaining lines will be plotted with
                           the default line width.

Axes parameters:

            'Legends' - specify a cell array of strings to be used for
                        the plot legends. If a cell contains an empty
                        string, the default legend string is built.
                        If a single string 'off' is given instead of a
                        cell array, then the legends are all switched
                        off.

            'XLabels' - Specify the labels to be used on the x-axis. The
                        units are added from the data object 'xunits'
                        property.

            'YLabels' - Specify the labels to be used on the y-axis. The
                        units are added from the data object 'yunits'
                        property. If the object contains complex data,
                        you should specify two y-labels for that object.

   The following axis properties also work with the 'all' keyword as
   described above in the line properties section.

            'XScales' - Specify the scales to be used on the x-axes.

            'YScales' - Specify the scales to the used on the y-axes. If
                        an object contains complex data, you should
                        specify two y-labels for that object.

            'XRanges' - Specify the ranges to be displayed on the x-axes.

            'YRanges' - Specify the ranges to the displayed on the
                        y-axes.


VERSION:     $Id: iplot.m,v 1.25 2008/02/12 09:29:24 mauro Exp $

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 overide 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)


 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = iplot(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = iplot(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = iplot(ao,'Category')

 HISTORY: 22-12-07 M Hewitson
             Creation

%%%
```

- abs ABS overloads the Absolute value operator for Analysis objects.
- find FIND particular samples that satisfy the input query and return a new AO.
- get GET get ao properties.
- imag IMAG overloads the imaginary operator for Analysis objects.
- plot PLOT a simple plot of analysis objects.
- real REAL overloads the real operator for Analysis objects.
- set SET set an analysis object property.

This function is called by:

SUBFUNCTIONS ^

- function varargout = fs_plot(varargin)
- function varargout = xy_plot(varargin)
- function varargout = y_plot(varargin)
- function varargout = xyz_plot(varargin)
- function out = getDefaultPlist(varargin)

SOURCE CODE ^

```
function varargout = iplot(varargin)
% IPLOT provides an intelligent plotting tool for LTPDA.
%
%%%
```

```
% Line parameters:
%
%        The following properties take cell array values. If the length of
%        the cell array is shorter than the number of lines to plot, the
%        remaining lines will be plotted with the default options. If the
%        cell array is of length 2 and the first cell contains the string
%        'all', then the second cell is used to set the propery of all
%        lines.
%
%             'LineColors' - a cell array of color definitions for each line.
%
%             'LineStyles' - a cell array of line styles.
%
%             'Markers'    - a cell array of markers.
%
%             'LineWidths' - a cell array of line widths. If the length of the
%                            cell array is shorter than the number of lines to
%                            plot, the remaining lines will be plotted with
%                            the default line width.
%
% Axes parameters:
%
%             'Legends' - specify a cell array of strings to be used for
%                         the plot legends. If a cell contains an empty
%                         string, the default legend string is built.
%                         If a single string 'off' is given instead of a
%                         cell array, then the legends are all switched
%                         off.
%
%             'XLabels' - Specify the labels to be used on the x-axis. The
%                         units are added from the data object 'xunits'
%                         property.
%
%             'YLabels' - Specify the labels to be used on the y-axis. The
%                         units are added from the data object 'yunits'
%                         property. If the object contains complex data,
%                         you should specify two y-labels for that object.
%
%    The following axis properties also work with the 'all' keyword as
%    described above in the line properties section.
%
%             'XScales' - Specify the scales to be used on the x-axes.
%
%             'YScales' - Specify the scales to the used on the y-axes. If
%                         an object contains complex data, you should
%                         specify two y-labels for that object.
%
%             'XRanges' - Specify the ranges to be displayed on the x-axes.
%
%             'YRanges' - Specify the ranges to the displayed on the
%                         y-axes.
%
%
```

```
% VERSION:     $Id: iplot.m,v 1.25 2008/02/12 09:29:24 mauro Exp $
%
% 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 overide 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)
%
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = iplot(ao, 'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = iplot(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = iplot(ao,'Category')
%
% 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.
%

%% Check if this is a call for parameters, version or category

VERSION = '$Id: iplot.m,v 1.25 2008/02/12 09:29:24 mauro Exp $';
CATEGORY = 'Output';
bs      = [];

if  (nargin == 2 || nargin == 3) &&...
    isa(varargin{1}, 'ao') &&...
    ischar(varargin{2})
  if strcmp(varargin{2}, 'Params')
    if nargin == 2
      varargout{1} = getDefaultPlist();
    else
      varargout{1} = getDefaultPlist(varargin{3});
    end
    return
  elseif strcmp(varargin{2}, 'Version')
    varargout{1} = VERSION;
    return
  elseif strcmp(varargin{2}, '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, upl, invars] = collect_inputs(varargin, 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

% get default plist
dpl = getDefaultPlist('tsdata');
% 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];

%----------- XYDATA

% get default plist
dpl = getDefaultPlist('xydata');
% 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];

%----------- XYZDATA

% get default plist
dpl = getDefaultPlist('xyzdata');
% 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];

%----------- CDATA

% get default plist
dpl = getDefaultPlist('cdata');
% 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];

%----------- FSDATA

% get default plist
dpl = getDefaultPlist('fsdata');
% 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];

%% 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


%-------------------------------------------------------------------------
% 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');

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
    a = aos(j);
    y = a.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 = [];

    % Get this AO
    a = aos(j);

    %------- Apply math functions
    x = a.data.x;
    y = a.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   = plot(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 = plot(tsax_r, x, y);
          li = [li plot(tsax_i, x, imag(y))];
          tsli = [tsli li];
          ylabelr = 'real';
          ylabeli = 'imag';
          yscaleR = 'lin';
          yscaleI = 'lin';
          xscaleR = 'lin';
          xscaleI = 'lin';
        case 'absdeg'
          li   = plot(tsax_r, x, abs(y));
          li   = [li plot(tsax_i, x, ltpda_phase(y))];
          tsli = [tsli li];
          ylabelr = 'amplitude';
          ylabeli = 'Phase [deg]';
          yscaleR = 'log';
          yscaleI = 'lin';
          xscaleR = 'log';
          xscaleI = 'log';
        case 'absrad'
          li   = plot(tsax_r, x, abs(y));
```

```
          li   = [li plot(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 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', a.data.yunits));
      ylstrR = [ylstrR ' [' ymath ']' ];
    else
      ylstrR = [ylstrR ' [' a.data.yunits ']' ];
    end
    ylabel(tsax_r, ylstrR);
    if ishandle(tsax_i)
      ylabel(tsax_i, ylstrI);
    end

    % 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', a.data.xunits));
      xlstr = [xlstr ' [' xmath ']' ];
    else
      xlstr = [xlstr ' [' a.data.xunits ']' ];
    end
    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 = ltpda_label(a.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, legendStr{end});
      end
    end

  end % End loop over AOs

  % Process legends for stacked plots
  if legendsOn
    if strcmp(arrangement, 'stacked')
      h = legend(tsax_r, legendStr);
      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

%-----------------------------------------------------------------------
% 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, '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');

% 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 j=1:Na
      % Get this AO
      a = aos(j);
      if isa(a.data, 'tsdata')
        if a.data.t0.utc_epoch_milli/1000 < T0
          T0 = floor(a.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
    a = aos(j);
    toff = 0;

    % what figures do we need?
    switch arrangement
      case 'single'

        tsfig = [tsfig figure];
        tsax = subplot(1,1,1);
        col = colors{1};
        if isa(a.data, 'tsdata')
```

```
          torigin = a.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(a.data, 'tsdata')
          toff = a.data.t0.utc_epoch_milli/1000 - T0;
        else
          toff = 0;
        end
        if isa(a.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(a.data, 'tsdata')
          torigin = a.data.t0;
        end
      otherwise
        error('### Unknown plot arrangement');
    end

    %------- Apply math functions

    % need t0 offset for this time-series
    x = a.data.x + toff;
    y = a.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   = plot(tsax(end), x, y);
```

```
    tsli = [tsli li];
    if isa(a.data, 'tsdata')
      title(sprintf('Time origin: %s', torigin.time_str));
    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', a.data.yunits));
      ylstr = [ylstr ' [' ymath ']' ];
    else
      ylstr = [ylstr ' [' a.data.yunits ']' ];
    end
    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', a.data.xunits));
      xlstr = [xlstr ' [' xmath ']' ];
    else
      xlstr = [xlstr ' [' a.data.xunits ']' ];
    end
    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 = ltpda_label(a.name);
      end
    end
    legendStr = [legendStr cellstr(lstr)];

    % Set the legend now if we can
    if legendsOn
      if strcmp(arrangement, 'single') || strcmp(arrangement, 'subplots')
        legend(legendStr{end});
      end
    end
  end

  % Process legends for stacked plots
  if legendsOn
    if strcmp(arrangement, 'stacked')
      h = legend(legendStr);
      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


%-----------------------------------------------------------------------
% 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');
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

    % Get this AO
    a = aos(j);

    % 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(a.data.y)
      x = 1:length(a.data.y);
      y = a.data.y;
    else
      x = real(a.data.y);
      y = imag(a.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 = plot(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', a.data.yunits));
      ylstr = [ylstr ' [' ymath ']' ];
    else
      ylstr = [ylstr ' [' a.data.yunits ']' ];
    end
    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', a.data.xunits));
      xlstr = [xlstr ' [' xmath ']' ];
    else
      xlstr = [xlstr ' [' a.data.xunits ']' ];
    end
    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 = ltpda_label(a.name);
        legendStr = [legendStr cellstr(lstr)];
      end
    end

    % Set the legend now if we can
    if legendsOn
      if strcmp(arrangement, 'single') || strcmp(arrangement, 'subplots')
        legend(legendStr{end});
      end
    end
  end

  % Process legends for stacked plots
  if legendsOn
    if strcmp(arrangement, 'stacked')
      h = legend(legendStr);
      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

%-----------------------------------------------------------------------
% 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');

% 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  = [];

% Legend holder
legendStr = [];

if ~isempty(aos)

  % Now loop over AOs
  Na = length(aos);
  for j=1:Na

    % Get this AO
    a = aos(j);

    % what figures do we need?
    switch arrangement
      case 'single'

        tsfig = [tsfig figure];
        tsax = subplot(1,1,1);

      case 'subplots'

        if j == 1, tsfig = figure; end
        tsax = [tsax subplot(Na, 1, j)];

      otherwise
```

```
        error('### Unknown plot arrangement');
    end

    %------- Apply math functions
    x = a.data.x;
    y = a.data.y;
    z = a.data.z;


    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
    zmath = '';
    if j<=length(zmaths)
      if ~isempty(zmaths{j})
        eval(sprintf('z = %s;', zmaths{j}));
        zmath = zmaths{j};
      end
    end

    %------- Plot the data

    idcs = pcolor(x,y,z);
    tsli = [tsli idcs(1:end).'];

    % plot properties
    set(idcs, 'EdgeColor', 'none');

    %------- Axis properties

    % Reverse y-direction for spectrograms
    set(tsax(end), 'YDir', 'reverse');

    % 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', a.data.yunits));
```

```
      ylstr = [ylstr ' [' ymath ']' ];
    else
      ylstr = [ylstr ' [' a.data.yunits ']' ];
    end
    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', a.data.xunits));
      xlstr = [xlstr ' [' xmath ']' ];
    else
      xlstr = [xlstr ' [' a.data.xunits ']' ];
    end
    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 = ltpda_label(a.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


%-------------------------------------------------------------------------
% Default Parameter Lists
%
function out = getDefaultPlist(varargin)

% list of available parameter sets
sets = {'tsdata', 'fsdata', 'cdata', 'xydata'};

% Get the LTPDA color set for lines
colors = getappdata(0,'ltpda_default_plot_colors');

if nargin == 0
  out = sets;
  return
end

set = varargin{1};

out = plist('Colors', colors, ...
  'Arrangement', 'stacked');

switch set
  case 'fsdata'
    out = append(out, 'type', 'fsdata', ...
                      'complexPlotType', 'absdeg', ...
                      'XLabel', 'Frequency');
  case 'tsdata'
    out = append(out, 'type', 'tsdata', ...
                      'Xlabel', 'Time', ...
                      'Ylabel', 'Amplitude');
  case 'xydata'
```

```
    out = append(out, 'type', 'xydata', ...
                      'Xlabel', 'X-data', ...
                      'Ylabel', 'Y-data', ...
                      'YMaths', '', ...
                      'XMaths', '');
  case 'xyzdata'
    out = plist('Colors', colors, 'Arrangement', 'single', ...
                      'type', 'xydata', ...
                      'Xlabel', 'Time', ...
                      'Ylabel', 'Frequency',...
                      'Zlabel', 'Amplitude', ...
                      'YMaths', '', ...
                      'ZMaths', '', ...
                      'XMaths', '');
  case 'cdata'
    out = append(out, 'type', 'tsdata', ...
                      'Xlabel', 'Index', ...
                      'Ylabel', 'Value');
  otherwise
    out = plist();
end



%% OLD BELOW HERE
% function varargout = iplot(varargin)
% % PLOT plots an array of analysis objects.
% %
% %%%%
```

```
%
% VERSION = '$Id: iplot.m,v 1.25 2008/02/12 09:29:24 mauro Exp $';
%
%   %%%%%%%%%%   Check if this is a call for parameters   %%%%%%%%%%
%   if nargin == 2
%     if isa(varargin{1}, 'ao') && ischar(varargin{2})
%       in = char(varargin{2});
%       if strcmp(in, 'Params')
%         varargout{1} = getDefaultPL();
%       elseif strcmp(in, 'Version')
%         varargout{1} = VERSION;
%         return
%       end
%     end
%   end
%
%   axes_handle   = [];
%   ao_s          = [];
%   pl_s          = [];
%   line_spec     = [];
%   prop_val      = {};
%
%   %%%%%%%%%%   Check input values   %%%%%%%%%%
%   var_argin = varargin;
%
%   while ~isempty(var_argin)
%     if ishandle(var_argin{1}) & isempty(axes_handle)
%       axes_handle = var_argin{1};
%       var_argin = var_argin(2:end);
%     elseif isa(var_argin{1}, 'ao')
%       var_argin{1} = reshape(var_argin{1}, 1, []);
%       ao_s = [ao_s var_argin{1}];
%       var_argin = var_argin(2:end);
%     elseif isa(var_argin{1}, 'plist')
%       pl_s = [pl_s var_argin{1}];
%       var_argin = var_argin(2:end);
%     elseif ischar(var_argin{1}) && islinespec(var_argin{1}) && isempty
(line_spec)
%       line_spec = var_argin{1};
%       var_argin = var_argin(2:end);
%     else
%       if length(var_argin) == 1
%         error('### There is only one propery [%s] without value left',
var_argin{1});
%       elseif length(var_argin) >= 2
%         prop = var_argin{1};
%         val  = var_argin{2};
%         if ismember(prop, {'complex_view', ...
%                            'title_str',    ...
%                            'legend_names', ...
%                            'xlabel_des',   ...
%                            'ylabel_des'})
%           pl_s = [pl_s plist(prop, val)];
```

```
%         else
%           prop_val = [prop_val, prop, val];
%         end
%         var_argin = var_argin(3:end);
%       end
%     end
%   end
%
%   pl_s = combine(pl_s, getDefaultPL());
%
%   %%%%%%%%%  Get default values from the application-defined data.   %%%%%%%%
%
%   len_legend  = getappdata(0, 'wraplegendstringat');
%
%   %%%%%%%%%
```

```
%     sort_ao.(class_name).legend_names = find(pl_s, 'legend_names');
%     sort_ao.(class_name).xlabel_des   = find(pl_s, 'xlabel_des');
%     sort_ao.(class_name).ylabel_des   = find(pl_s, 'ylabel_des');
%
%     if ~isreal(y)
%       sort_ao.(class_name).complex = true;
%     end
%     if (max(x)-min(x)) / 10^floor(log10(mean(abs(x)))) > 9
%       sort_ao.(class_name).xscale = 'log';
%     end
%     if min(x) < 0
%       sort_ao.(class_name).xscale = 'linear';
%     end
%     if (max(y)-min(y)) / 10^floor(log10(max(mean(abs(y))))) > 9
%       sort_ao.(class_name).yscale = 'log';
%     end
%     if min(y) < 0
%       sort_ao.(class_name).yscale = 'linear';
%     end
%     if strcmp(class_name, 'tsdata')
%       sort_ao.(class_name).xscale = 'linear';
%     end
%
%
%     %%%%%%%%%%   Spezial fields for the different data objects   %%%%%%%%%%
%
%     %%%%%   FSDATA   %%%%%
%     if strcmp(class_name, 'fsdata')
%       if length (ao_s(ii).data.enbw) > 1
%         sort_ao.(class_name).enbw = find(pl_s, 'enbw');
%       end
%
%       % For fsdata set the x-axis and the y-axis to a logarithmic scale
%       sort_ao.(class_name).xscale = 'log';
%       sort_ao.(class_name).yscale = 'log';
%     end
%
%     %%%%%   TSDATA   %%%%%
%     if strcmp(class_name, 'tsdata')
%
%       t0 = ao_s(ii).data.t0.utc_epoch_milli/1000;
%       if t0 < T0_off
%         T0_off = t0;
%         sort_ao.(class_name).T0_off = t0;
%       end
%     end
%
%   end
%
%   %%%%%%%%%%
```

```
%
%   out_line_h   = [];
%   out_axes_h   = [];
%   out_figure_h = [];
%
%   for ii = 1:length(fields)
%     class_name = fields{ii};
%
%     %%%%%%%%%%   Add the t0 offset to all aos with tsdata objects.   %%%%%%%%%%
%     if strcmp(class_name, 'tsdata')
%       for jj = 1:length(sort_ao.(class_name).ao)
%         curr_ao = sort_ao.(class_name).ao(jj);
%
%         t0 = curr_ao.data.t0.utc_epoch_milli/1000;
%         curr_ao.data.x = curr_ao.data.x + t0 - sort_ao.(class_name).T0_off;
%
%         sort_ao.(class_name).ao(jj) = curr_ao;
%       end
%     end
%
%     %%%%%%%%%%   Set default values   %%%%%%%%%%
%     % Set default title
%     if isempty (sort_ao.(class_name).title_str)
%       if sort_ao.(class_name).complex == true
%         add_text = 'complex ';
%       else
%         add_text = '';
%       end
%       switch (class_name)
%         case 'fsdata'
%           sort_ao.(class_name).title_str = sprintf('Plot of %sfrequency series AOs', add_text);
%         case 'tsdata'
%           sort_ao.(class_name).title_str = sprintf('Plot of %stime series AOs', add_text);
%         case 'cdata'
%           sort_ao.(class_name).title_str = sprintf('Plot of %sconstant data AOs', add_text);
%         case 'xydata'
%           sort_ao.(class_name).title_str = sprintf('Plot of %sxy data AOs', add_text);
%         otherwise
%           error('### Unknown data object ''%s''', class_name);
%       end
%     end
%
%     % Set default x-, y-units
%     if isempty(sort_ao.(class_name).ao(1).data.xunits)
%       xunits = 'unknown';
%     else
%       xunits= sort_ao.(class_name).ao(1).data.xunits;
%     end
```

```
%     if isempty(sort_ao.(class_name).ao(1).data.yunits)
%       yunits = 'unknown';
%     else
%       yunits= sort_ao.(class_name).ao(1).data.yunits;
%     end
%
%     fig_h = gcf;
%
%     %%%%%%%%%%   Plot complex values   %%%%%%%%%%
%     if sort_ao.(class_name).complex == true
%
%       if strcmp(sort_ao.(class_name).complex_view, 're/im')
%         axes_h1 = subplot(2,1,1);
%         line_h1 = plot(axes_h1, real(sort_ao.(class_name).ao), line_spec, prop_val{:});
%         axes_h2 = subplot(2,1,2);
%         line_h2 = plot(axes_h2, imag(sort_ao.(class_name).ao), line_spec, prop_val{:});
%
%         label_x1 = [sort_ao.(class_name).xlabel_des ' [' xunits ']'];
%         label_y1 = [sort_ao.(class_name).ylabel_des 'Real [' yunits ']'];
%         label_y2 = [sort_ao.(class_name).ylabel_des 'Imaginary [' yunits ']'];
%
%         % Do not set the xscale if it is set as a propertie/value pair
%         if ~ismember('xscale', lower(prop_val))
%           % If the 'key' exist in the parameter list then do NOT replace the
%           % value. (see combine) If the user use a different syntax (he use
%           % upper case letters) this value is put before the user value.
%           % This function take ALWAYS the user value.
%           pl_s = combine(pl_s, plist('xscale', sort_ao.(class_name).xscale));
%         end
%
%         % Plot both y-axes as a linear scale
%         pl_s = pset(pl_s, 'Yscale', 'linear');
%
%       else % if strcmp(sort_ao.(class_name).complex_view, 'mag/deg')
%         axes_h1 = subplot(3,1,1:2);
%         line_h1 = plot(axes_h1, abs(sort_ao.(class_name).ao), line_spec, prop_val{:});
%         axes_h2 = subplot(3,1,3);
%         line_h2 = plot(axes_h2, phase(sort_ao.(class_name).ao), line_spec, prop_val{:});
%
%         label_x1 = [sort_ao.(class_name).xlabel_des ' [' xunits ']'];
%         label_y1 = [sort_ao.(class_name).ylabel_des 'Magnitude [' yunits ']'];
%         label_y2 = [sort_ao.(class_name).ylabel_des 'Phase [' yunits ']'];
%
%         % Do not set the xscale if it is set as a propertie/value pair
%         if ~ismember('xscale', lower(prop_val))
%           % If the 'key' exist in the parameter list then do NOT replace the
%           % value. (see combine) If the user use a different syntax (he use
%           % upper case letters) this value is put before the user value.
%           % This function take ALWAYS the user value.
```

```
%           pl_s = combine(pl_s, plist('xscale', sort_ao.(class_name).xscale));
%         end
%
%         % Do not set the yscale if it is set as a propertie/value pair
%         if ~ismember('yscale', lower(prop_val))
%           % If the 'key' exist in the parameter list then do NOT replace the
%           % value. (see combine) If the user use a different syntax (he use
%           % upper case letters) this value is put before the user value.
%           % This function take ALWAYS the user value.
%           pl_s = combine(pl_s, plist('yscale', sort_ao.(class_name).yscale));
%         end
%
%         % Plot always the second y-axes as a linear scale
%         pl_s = pset(pl_s, 'Yscale2', 'linear');
%       end
%
%       % Set x-, y-label
%       xlabel(axes_h2, label_x1);
%       ylabel(axes_h1, label_y1);
%       ylabel(axes_h2, label_y2);
%
%       out_line_h   = [out_line_h,   line_h1, line_h2];
%       out_axes_h   = [out_axes_h,   axes_h1, axes_h2];
%       out_figure_h = [out_figure_h, fig_h];
%
%     %%%%%%%%%%   Plot real values   %%%%%%%%%%
%     else
%
%       [line_h1, axes_h1] = plot(real(sort_ao.(class_name).ao), line_spec,
prop_val{:});
%
%       axes_h2 = axes_h1;
%       line_h2 = line_h1;
%
%       % Set x-, y-label
%       xlabel(axes_h1, [sort_ao.(class_name).xlabel_des ' [' xunits ']']);
%       ylabel(axes_h1, [sort_ao.(class_name).ylabel_des ' [' yunits ']']);
%
%       out_line_h   = [out_line_h,   line_h1];
%       out_axes_h   = [out_axes_h,   axes_h1];
%       out_figure_h = [out_figure_h, fig_h];
%
%       % Do not set the xscale if it is set as a propertie/value pair
%       if ~ismember('xscale', lower(prop_val))
%         % If the 'key' exist in the parameter list then do NOT replace the
%         % value. (see combine) If the user use a different syntax (he use
%         % upper case letters) this value is put before the user value.
%         % This function take ALWAYS the user value.
%         pl_s = combine(pl_s, plist('xscale', sort_ao.(class_name).xscale));
%       end
%
%       % Do not set the yscale if it is set as a propertie/value pair
%       if ~ismember('yscale', lower(prop_val))
```

```
%         % If the 'key' exist in the parameter list then do NOT replace the
%         % value. (see combine) If the user use a different syntax (he use
%         % upper case letters) this value is put before the user value.
%         % This function take ALWAYS the user value.
%         pl_s = combine(pl_s, plist('yscale', sort_ao.(class_name).yscale));
%       end
%
%     end % if complex == true
%
%     %%%%%%%%%%   Set Title     %%%%%%%%%%
%     title (axes_h1, sort_ao.(class_name).title_str);
%
%     %%%%%%%%%%   Set Title     %%%%%%%%%%
%     grid(axes_h1, 'on');
%     grid(axes_h2, 'on');
%
%     %%%%%%%%%%   Set legend names   %%%%%%%%%%
%     % Use the ao names for the legend
%     if isempty(sort_ao.(class_name).legend_names)
%       for jj = 1:length(sort_ao.(class_name).ao)
%
%         display_name = sort_ao.(class_name).ao(jj).name;
%         display_name = ltpda_label(display_name);
%         display_name = wrapLegendString(sprintf('%s', ltpda_label
(display_name)), len_legend);
%         set(line_h1(jj), 'DisplayName', ltpda_label(display_name))
%       end
%       legend(axes_h1, 'toggle')
%       % Use the legend names from the plist
%     else
%       legend(axes_h1, sort_ao.(class_name).legend_names);
%     end
%
%     %%%%%%%%%%   Set axes OR lineseries properties from the plist   %%%%%%%%%%
%     for jj = 1:length(pl_s.params)
%       p = pl_s.params(jj);
%
%       [idx_b, idx_e, dummy, match] = regexp(p.key, '\d', 'once');
%       p.key = strrep(p.key, match, '');
%
%       % Check key to the axes property
%       if isprop(line_h1, p.key)
%         if strcmp(match, '1')
%           set(line_h1, p.key, p.val);
%         elseif strcmp(match, '2')
%           set(line_h2, p.key, p.val);
%         else
%           set(line_h1, p.key, p.val);
%           set(line_h2, p.key, p.val);
%         end
%       % Check key to the axes property
%       elseif isprop(axes_h1, p.key)
%         if strcmp(match, '1')
```

```
%           set(axes_h1, p.key, p.val);
%         elseif strcmp(match, '2')
%           set(axes_h2, p.key, p.val);
%         else
%           set(axes_h1, p.key, p.val);
%           set(axes_h2, p.key, p.val);
%         end
%       end
%
%     end
%
%   end
%
%
%   %%%%%%%%%%   varargout{1} = line   handle   %%%%%%%%%%
%   %%%%%%%%%%   varargout{2} = axes   handle   %%%%%%%%%%
%   %%%%%%%%%%   varargout{3} = figure handle   %%%%%%%%%%
%
%   if nargout == 0
%   elseif nargout == 1
%     varargout{1} = out_line_h;
%   elseif nargout == 2
%     varargout{1} = out_line_h;
%     varargout{2} = out_axes_h;
%   elseif nargout == 3
%     varargout{1} = out_line_h;
%     varargout{2} = out_axes_h;
%     varargout{3} = out_figure_h;
%   else
%     error('### Unknown number of outputs.')
%   end
%
%
% end % function varargout = iplot(varargin)
%
% %%%%%%%%%%
```

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

isvalid

PURPOSE ^

ISVALID tests if the given ao has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given ao has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given ao has all the correct fields of the
              correct type.

 CALL:        isvalid(a);

              Returns 1 is a is a valid AO; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(ao, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(ao, 'Version')

              The following call returns the category:
              >> ver = isvalid(ao, 'Category')

 VERSION:     $Id: isvalid.m,v 1.6 2008/02/23 21:39:17 hewitson Exp $

 HISTORY: 25-09-07 M Hewitson
              Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given ao has all the correct fields of the correct type.
%
%%%
```

```
  if isa(varargin{j}, 'ao')
    as = [as varargin{j}];
  end
end

res = zeros(size(as));
for j=1:numel(as)
  a = as(j);
  if ischar(a.name)                    && ...
       (isa(a.data, 'cdata')    || ...
        isa(a.data, 'fsdata')   || ...
        isa(a.data, 'tsdata')   || ...
        isa(a.data, 'xydata')   || ...
        isa(a.data, 'xyzdata')  || ...
        isempty(a.data))                && ...
     isa(a.hist, 'history')             && ...
     isa(a.provenance, 'provenance')    && ...
     ischar(a.description)              && ...
     ischar(a.mfile)                    && ...
     ischar(a.mfilename)                && ...
     ischar(a.mdlfile)                  && ...
     ischar(a.mdlfilename)              && ...
       (isa(a.plist, 'plist')   || ...
        isempty(a.plist))               && ...
     ischar(a.version)                  && ...
     isa(a.created, 'time')

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

le

PURPOSE ^

LE overloads <= operator for analysis objects. Compare the y-axis values.

SYNOPSIS ^

function a = le(b, c, 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<=;

 POSSIBLE VALUES: b: Analysis object
                  c: Analysis object or a scalar.

 VERSION: $Id: le.m,v 1.8 2008/02/25 18:04:22 ingo Exp $

 HISTORY: 06-02-2007 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.

This function is called by:

SOURCE CODE ^

```
function a = le(b, c, varargin)
% LE overloads <= operator for analysis objects. Compare the y-axis values.
%
```

```
%%%
```

```
a = x<=y;

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

len

PURPOSE ^

LEN overloads the length operator for Analysis objects. Length of the data samples.

SYNOPSIS ^

function data_len = 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);

 POSSIBLE VALUES: ao_in  = [ao2 ao3]
                  ao_in  = ao_vector
                  ao_in  = ao_matrix

 VERSION: $Id: len.m,v 1.10 2008/02/12 19:29:33 hewitson Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = len(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = len(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = len(ao,'Category')

 HISTORY: 24-04-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.

This function is called by:

- downsample DOWNSAMPLE AOs containing time-series data.

SUBFUNCTIONS ^

- function pl_default = getDefaultPL()

SOURCE CODE ^

```
function data_len = len(varargin)
% LEN overloads the length operator for Analysis objects. Length of the data samples.
%
%%%
```

```
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ln

PURPOSE ^

LN overloads the log operator for Analysis objects. Natural logarithm.

SYNOPSIS ^

function ao_out = ln (varargin)

DESCRIPTION ^

```
 LN overloads the log operator for Analysis objects. Natural logarithm.

%%%
```

```
 HISTORY: 23-05-2007 Diepholz
             Creation

%%%
```

```
% >> version = ln(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = ln(ao,'Category')
%
% HISTORY: 23-05-2007 Diepholz
%             Creation
%
%%%
```

```

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

lt

PURPOSE ^

LT overloads < operator for analysis objects. Compare the y-axis values.

SYNOPSIS ^

function a = lt(b, c, varargin)

DESCRIPTION ^

```
 LT overloads < operator for analysis objects. Compare the y-axis values.

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

 DESCRIPTION: LT overloads < operator for analysis objects.
              Compare the y-axis values.

 CALL: a = b<c;

 POSSIBLE VALUES: b: Analysis object
                  c: Analysis object or a scalar.

 VERSION: $Id: lt.m,v 1.8 2008/02/25 18:04:22 ingo Exp $

 HISTORY: 06-02-2007 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.

This function is called by:

SOURCE CODE ^

```
function a = lt(b, c, varargin)
% LT overloads < operator for analysis objects. Compare the y-axis values.
%
```

```
%%%
```

```
a = x<y;


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

md5

PURPOSE ^

MD5 computes an MD5 checksum from an analysis objects.

SYNOPSIS ^

function h = md5(varargin)

DESCRIPTION ^

```
 MD5 computes an MD5 checksum from an analysis objects.

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

 DESCRIPTION: MD5 computes an MD5 checksum from an analysis objects.

 CALL:        h = md5(a)

 INPUTS:      a - input analysis object

 OUTPUTS:     h - md5 hash

 VERSION:     $Id: md5.m,v 1.5 2008/02/19 19:45:58 hewitson Exp $

 HISTORY:     15-09-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function h = md5(varargin)
% MD5 computes an MD5 checksum from an analysis objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
% DESCRIPTION: MD5 computes an MD5 checksum from an analysis objects.
%
% CALL:        h = md5(a)
%
% INPUTS:      a - input analysis object
%
% OUTPUTS:     h - md5 hash
%
% VERSION:     $Id: md5.m,v 1.5 2008/02/19 19:45:58 hewitson Exp $
%
% HISTORY:     15-09-2007 M Hewitson
%                 Creation
%
%%%
```

```
  h = [h cellstr(ltpda_hash(xmlwrite(xml), 'MD5'))];
end

h = reshape(h, size(as));

if numel(h) == 1
  h = cell2mat(h);
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

mux

PURPOSE ^

MUX concatonate AOs into a vector.

SYNOPSIS ^

function b = mux(varargin)

DESCRIPTION ^

```
 MUX concatonate AOs into a vector.

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

 DESCRIPTION: MUX concatenate AOs into a vector.

 CALL:        ao_vec = mux(a1 ,a2);

 VERSION:     $Id: mux.m,v 1.6 2008/02/12 09:29:24 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = mux(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = mux(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = mux(ao,'Category')

 HISTORY: 27-03-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.

This function is called by:

SUBFUNCTIONS ^

- function pl_default = getDefaultPL()

SOURCE CODE ^

```
function b = mux(varargin)
% MUX concatonate AOs into a vector.
%
%%%
```

```
    elseif strcmp(in, 'Category')
      b = CATEGORY;
      return
    end
  end
end

b = [];
for j=1:nargin
  if isa(varargin{j}, 'ao')
    b = [b varargin{j}];
  end
end

%% Get default params
function pl_default = getDefaultPL()

  pl_default = plist();
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for analysis objects.

SYNOPSIS ^

function result = ne(c1,c2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for analysis objects.

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

 DESCRIPTION: NE overloads the ~= operator for analysis objects.

 CALL:        result = ne(c1,c2)
              result = ne(c1,c2,  exc_list)
              result = ne(c1,c2, 'property1', 'property2')
              result = ne(c1,c2, 'class/property', 'class/property')
              result = ne(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = ne(c1,c2, 'name', 'created')
              result = ne(c1,c2, 'ao/name')

 INPUTS:      c1,c2    - input analysis objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two analysis objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.8 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.
- eq EQ overloads the == operator for analysis objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(c1,c2, varargin)
% NE overloads the ~= operator for analysis objects.
%
%%%
```

```
      return
    elseif strcmp(in, 'Version')
      result = VERSION;
      return
    elseif strcmp(in, 'Category')
      result = CATEGORY;
      return
    end
  end
end

if eq(c1,c2, varargin{:})
  result = 0;
else
  result = 1;
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

plot

PURPOSE ^

PLOT a simple plot of analysis objects.

SYNOPSIS ^

function varargout = plot(varargin)

DESCRIPTION ^

```
 PLOT a simple plot of analysis objects.

%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.
- get GET get ao properties.
- imag IMAG overloads the imaginary operator for Analysis objects.
- plot PLOT a simple plot of analysis objects.
- real REAL overloads the real operator for Analysis objects.
- set SET set an analysis object property.

This function is called by:

- iplot IPLOT provides an intelligent plotting tool for LTPDA.
- plot PLOT a simple plot of analysis objects.

SOURCE CODE ^

```
function varargout = plot(varargin)
% PLOT a simple plot of analysis objects.
%
%%%
```

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

```
        val  = var_argin{2};
        prop_val  = [prop_val, prop, val];
        var_argin = var_argin(3:end);
      end
    end
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %             For all ao_s             %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  for ii = 1:length(ao_s)

    %%%%%%%%%%   Setthe line colors   %%%%%%%%%%
    line_colors = getappdata(0, 'ltpda_default_plot_colors');

    %%% Set only the color if the color is not set in the linespec
    [res, style] = islinespec(line_spec);

    if isempty(style{3})
      prop_val{end+1} = 'Color';
      prop_val{end+1} = line_colors{ii};
    end

    %%%%%%%%%%   Get the x-, y-values   %%%%%%%%%%
    [x,y] = get_xy_values(ao_s(ii).data);

    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 % function varargout = plot(varargin)
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

save

PURPOSE ^

SAVE overloads save operator for analysis objects.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE overloads save operator for analysis objects.

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

 DESCRIPTION: SAVE overloads save operator for analysis objects.

 CALL:        Save an analysis object as a .mat file.
                >> save(a, 'blah.mat')
              Save an analysis object as an XML file.
                >> save(a, 'blah.xml')

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.16 2008/02/13 10:13:53 ingo Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = save(ao,'Category')

 HISTORY: 14-02-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.

This function is called by:

- export EXPORT export an analysis object to a text file.

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE overloads save operator for analysis objects.
%
%%%
```

```
VERSION  = '$Id: save.m,v 1.16 2008/02/13 10:13:53 ingo Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'ao') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

stairs

PURPOSE ^

STAIRS overloads the stairs function for Analysis Objects.

SYNOPSIS ^

function varargout = stairs(varargin)

DESCRIPTION ^

```
 STAIRS overloads the stairs function for Analysis Objects.

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

 DESCRIPTION: STAIRS overloads the stairs function for Analysis Objects.

 CALL:            stairs(a)
             h = stairs(a)

 INPUTS:      a  - analysis object(s)

 OTPUTS:      h  - plot handles

 PARAMETERS:  none

              The following call returns a parameter list object that
              contains the default parameter values:

              >> pl = stairs(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: stairs.m,v 1.8 2008/02/12 10:11:13 mauro Exp $

 HISTORY: 02-03-07 M Hewitson
            Creation.
```

```
%%%
```

```
%
%              >> category = foo(ao,'Category')
%
% VERSION:     $Id: stairs.m,v 1.8 2008/02/12 10:11:13 mauro Exp $
%
% HISTORY: 02-03-07 M Hewitson
%            Creation.
%
%%%
```

```
      Nbins = varargin{j};
    end
  end
end

% produce one plist
if isa(ps, 'plist')
  pl = combine(ps, getDefaultPL());
else
  pl = getDefaultPL();
end

% If we have X from command input, override plist
if ~isempty(Xbins)
  pl = pset(pl, 'X', Xbins);
end

% If we have N from command input, override plist
if Nbins>0
  pl = pset(pl, 'N', Nbins);
end

% Get parameters
% N = find(pl, 'N');
% X = find(pl, 'X');

colors = getappdata(0, 'ltpda_default_plot_colors');

%---------------- Loop over input AOs

% Initialise output
handles = [];
% start looping
for j=1:numel(as)
  a = as(j);
  legendStr = [];

  % get data
  [x,y] = get_xy_values(a.data);
  if isempty(x)
    x = 1:length(y);
  end

  % make stair plot
  col = colors{mod(j-1,length(colors))+1};
  h = stairs(x,y);
  legendStr = [legendStr cellstr(sprintf('%s', ltpda_label(a.name)))];
  set(h, 'Color', col);
  title('Stair plot');
  grid on;
  legend(legendStr)
  ylabel(sprintf('Y [%s]', a.data.yunits));
```

```
  xlabel(sprintf('X [%s]', a.data.xunits));
  hold on;

  % store handles for output
  handles = [handles; h];

end

%---------------- Set outputs
if nargout == 1
  varargout{1} = handles;
end
%-------------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

plo = plist();
% plo = append(plo, 'N', '');
% plo = append(plo, 'X', '');


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input Analysis object(s).

SYNOPSIS ^

function cmd = 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)

 VERSION:     $Id: string.m,v 1.5 2008/02/12 19:29:33 hewitson Exp $

 HISTORY: 29-03-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- get GET get ao properties.
- hist HIST overloads the histogram function (hist) of MATLAB for Analysis Objects.
- 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).

SOURCE CODE ^

```
function cmd = string(varargin)
```

```
% STRING writes a command string that can be used to recreate the input Analysis object(s).
%
%%%
```

```
  plstr = string(pl);

  cmd = [cmd 'ao(' ...
             plstr ...
             ') '];
end

cmd = [cmd ']'];

if strcmp(cmd, '[]')
  cmd = '';
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for analysis objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for analysis objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for analysis objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.17 2008/02/22 22:26:41 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for analysis objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for analysis objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.17 2008/02/22 22:26:41 hewitson Exp $
%
% HISTORY:     31-01-07 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)

 VERSION:     $Id: timeshift.m,v 1.1 2008/02/24 19:52:00 hewitson Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = timeshift(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = timeshift(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = timeshift(ao,'Category')

 HISTORY: 23-02-08 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR overloads char() function for analysis objects.
- get GET get ao properties.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = timeshift(varargin)
% TIMESHIFT for AO/tsdata objects, shifts the time axis such that x(1) = 0.
%
%%%
```

```
    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
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, ps, invars] = collect_inputs(varargin, in_names);


% Check input analysis object
for j=1:numel(as)
  a     = as(j);
  d     = get(a, 'data');
  dinfo = whos('d');

  % Which data type do we have
  dtype = dinfo.class;
  switch dtype
    case 'tsdata'

      x0 = a.data.x(1);
      t0 = a.data.t0 - x0;
      x  = a.data.x - x0;
      a.data.x  = x;
      a.data.t0 = t0;

    case {'fsdata', 'cdata', 'xydata'}
      error('### I don''t work for frequency-series, xy and constant data.');
    otherwise
      error('### unknown data type. They can not be addded.')
  end

  % add to output
  bs = [bs a];
end

% Reshape the ouput to the same size of the input
bs = reshape(bs, size(as));

```

```
varargout{1} = bs;


%-----------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

plo = plist();

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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


 VERSION:     $Id: validate.m,v 1.1 2008/02/22 18:39:12 hewitson Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = validate(ao, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = validate(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = validate(ao,'Category')
```

```
 HISTORY: 22-02-08 M Hewitson
             Creation

%%%
```

```
%
% >> category = validate(ao,'Category')
%
% HISTORY: 22-02-08 M Hewitson
%             Creation
%
%%%
```

```

  % Initialise the match vector
  matches = zeros(length(nodes),1);

  % Loop over each history node, i.e., each function
  for jj=1:length(nodes)

    % This node
    node = nodes(jj);
    % Assume failure to start with
    matches(jj) = 0;

    % get fcn name
    fcnname = node.names;

    % Find all functions in MATLAB path with this name
    mfiles = which(fcnname, '-ALL');

    % Find all matches in versions{}
    idx = ismember(versions(:,2), fcnname);
    ltpdafiles = versions(find(idx==1), 1);

    % check against each one found
    for kk=1:length(mfiles)
      mfile = mfiles{kk};

      % make file hash
      try
        % Load the full file contents
        fd = fopen(mfile, 'r');
        fc = fscanf(fd, '%s');
        fclose(fd);
        % Make MD5 hash
        mhash = ltpda_hash(fc, 'MD5');

        % Check against all ltpda files with this function name
        for ll=1:length(ltpdafiles)
          % this file
          lfile = ltpdafiles{ll};
          % Load file contents
          fd = fopen(lfile, 'r');
          fc = fscanf(fd, '%s');
          fclose(fd);
          % Make MD5 hash
          lhash = ltpda_hash(fc, 'MD5');

          % Compares hashes
          if strcmp(mhash, lhash)
            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)
        disp(sprintf('!!!! 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;
    disp('*** AO validated')
  else
    passed(ec) = 0;
    disp('### AO not validated')
  end
end % end loop over all objects

% Set outputs
varargout{1} = passed;

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

< Master index

Index for classes/@ao/private >

Index for classes/@ao/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: ao/private %%%%%%%%%%%%%%%%%%%%%
aooperate	AOOPERATE implements specified operator overload for AOs.
retrieve	RETRIEVE analysis objects with given ids from the LTPDA repository specified by the input
xdoc2ao	XDOC2AO converts a DOM object to an AO.

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@cdata >

Index for classes/@cdata

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: cdata %%%%%%%%%%%%%%%%%%%%%
cdata	CDATA constant data object class constructor.
char	CHAR convert a cdata object into a string.
display	DISPLAY implement terminal display for cdata object.
eq	EQ overloads the == operator for cdata objects.
get	GET get csdata properties.
get_xy_values	GET_XY_VALUES returns the values in vals.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given cdata has all the correct fields of the correct type.
ne	NE overloads the ~= operator for cdata objects.
save	SAVE a cdata object to file.
set	SET set a cdata property.
set_xy_axis	SET_XY_AXIS set the x-axis and y-axis values.
single_operation	SINGLE_OPERATION implements specified operator overload for cdata objects.
subsasgn	SUBSASGN define index assignment for fsdata properties.
subsref	SUBSREF Define field name indexing for cdata objects.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@cdata/private >

Index for classes/@cdata/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%%%% class: cdata/private %%%%%%%%%%%%%%%%%%%%%%%
retrieve	RETRIEVE cdata objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@fsdata >

Index for classes/@fsdata

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: fsdata %%%%%%%%%%%%%%%%%%%%%
char	CHAR convert a param object into a string.
display	DISPLAY implement terminal display for fsdata object.
eq	EQ overloads the == operator for fsdata objects.
fsdata	FSDATA frequency-series object class constructor.
get	GET get a fsdata property.
get_xy_values	GET_XY_VALUES returns the x-axis and/or y-axis values.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given fsdata has all the correct fields of the correct type.
ne	NE overloads the ~= operator for fsdata objects.
reshapeF	RESHAPEF reshape the frequency vector to match the y vector.
save	SAVE an fsdata object to file.
set	SET set a fsdata property.
setFreq	SETFREQ sets the frequency vector of a fsdata object based on the length of the y vector and the sample rate.
set_xy_axis	SET_XY_AXIS set the x-axis and y-axis values.
single_operation	SINGLE_OPERATION implements specified operator overload for fsdata objects.
subsasgn	SUBSASGN define index assignment for fsdata properties.
subsref	SUBSREF Define field name indexing for fsdata objects.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@fsdata/private >

Index for classes/@fsdata/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%%%% class: fsdata/private %%%%%%%%%%%%%%%%%%%%%%%%
retrieve	RETRIEVE fsdata objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 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.
display	DISPLAY implement terminal display for history object.
eq	EQ overloads the == operator for history objects.
get	GET get history object properties.
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.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given history has all the correct fields of the correct type.
ne	NE overloads the ~= operator for history objects.
plot	PLOT plots a history object as a tree diagram.
save	SAVE a history object to file.
set	SET set a history property.
string	STRING writes a command string that can be used to recreate the input history object.
subsasgn	SUBSASGN define index assignment for history properties.
subsref	SUBSREF Define field name indexing for history objects.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@history/private >

Index for classes/@history/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%%%% class: history/private %%%%%%%%%%%%%%%%%%%%%%%%
retrieve	RETRIEVE history objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 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.
display	DISPLAY overloads display functionality for mfir objects.
eq	EQ overloads the == operator for mfir objects.
get	GET get mfir properties.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given mfir has all the correct fields of the correct type.
mfir	MFIR FIR filter object class constructor.
ne	NE overloads the ~= operator for mfir objects.
redesign	REDESIGN redesign the input filter to work for the given sample rate.
resp	RESP Make a frequency response of the filter.
save	SAVE an mfir object to file.
set	SET set a mfir object property.
string	STRING writes a command string that can be used to recreate the input filter object.
subsasgn	SUBSASGN define index assignment for mfir properties.
subsref	SUBSREF Define field name indexing for mfir objects.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index

Index for classes/@mfir/private >

Index for classes/@mfir/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: mfir/private %%%%%%%%%%%%%%%%%%%%%
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 mfir().
parseFilterParams	PARSEFILTERPARAMS parses the input plist and returns a full plist for
retrieve	RETRIEVE mfir objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 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.
display	DISPLAY overloads display functionality for miir objects.
eq	EQ overloads the == operator for miir objects.
get	GET get miir properties.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given miir has all the correct fields of the correct type.
miir	MIIR IIR filter object class constructor.
ne	NE overloads the ~= operator for miir objects.
redesign	REDESIGN redesign the input filter to work for the given sample rate.
resp	RESP Make a frequency response of the filter.
save	SAVE an miir object to file.
set	SET set a miir object property.
string	STRING writes a command string that can be used to recreate the input filter object.
subsasgn	SUBSASGN define index assignment for miir properties.
subsref	SUBSREF Define field name indexing for miir objects.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@miir/private >

Index for classes/@miir/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: miir/private %%%%%%%%%%%%%%%%%%%%
filload	Load a LISO *_iir.fil file to get the filter taps and return a
mfiltload	MFILTLOAD Load in a filter from *.filt files.
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
retrieve	RETRIEVE miir objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@param/private >

Index for classes/@param/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: param/private %%%%%%%%%%%%%%%%%%%%%%
retrieve	RETRIEVE param objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@plist/private >

Index for classes/@plist/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: plist/private %%%%%%%%%%%%%%%%%%%%
retrieve	RETRIEVE plist objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@pole/private >

Index for classes/@pole/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: pole/private %%%%%%%%%%%%%%%%%%%%%
pfq2ri	Convert frequency/Q pole/zero representation into real
pri2fq	Convert comlpex pole/zero into frequency/Q pole/zero representation.
retrieve	RETRIEVE pole objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 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.
display	DISPLAY overload terminal display for provenance objects.
eq	EQ overloads the == operator for provenance objects.
get	GET get provenance properties.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given provenance has all the correct fields of the correct type.
ne	NE overloads the ~= operator for provenance objects.
provenance	PROVENANCE constructors for provenance class.
save	SAVE a provenance object to file.
set	SET set an provenance property.
string	STRING writes a command string that can be used to recreate the input provenance object.
subsasgn	SUBSASGN define index assignment for provenance properties.
subsref	SUBSREF Define field name indexing for provenance objects.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@provenance/private >

Index for classes/@provenance/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%%%% class: provenance/private %%%%%%%%%%%%%%%%%%%%%%%%
retrieve	RETRIEVE provenance objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@pzmodel/private >

Index for classes/@pzmodel/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: pzmodel/private %%%%%%%%%%%%%%%%%%%%%%
abcascade	Cascade two filters together to get a new filter.
pzm2ab	PZM2AB convert pzmodel to IIR filter coefficients using bilinear
retrieve	RETRIEVE pzmodel objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@specwin/private >

Index for classes/@specwin/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: specwin/private %%%%%%%%%%%%%%%%%%%%%%
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
retrieve	RETRIEVE specwin objects with given ids from the LTPDA repository specified by the input
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 Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< 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.
display	DISPLAY overloads display functionality for time objects.
eq	EQ overloads the == operator for time objects.
format	FORMAT Returns the time in the handover format.
get	GET get time properties.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given time has all the correct fields of the correct type.
minus	MINUS overloads - operator for time objects.
ne	NE overloads the ~= operator for time objects.
plus	PLUS overloads - operator for time objects.
save	SAVE a time object to file.
set	SET set time properties.
string	STRING writes a command string that can be used to recreate the input time object.
subsasgn	SUBSASGN define index assignment for time properties.
subsref	SUBSREF Define field name indexing for time objects.
time	TIME time object class constructor.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@time/private >

Index for classes/@time/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%%%% class: time/private %%%%%%%%%%%%%%%%%%%%%%
retrieve	RETRIEVE time objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@timeformat >

Index for classes/@timeformat

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: timeformat %%%%%%%%%%%%%%%%%%%%%
char	CHAR convert a timeformat object into a string.
display	DISPLAY overloads display functionality for timeformat objects.
eq	EQ overloads the == operator for timeformat objects.
get	GET get timeformat properties.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given timeformat has all the correct fields of the correct type.
ne	NE overloads the ~= operator for timeformat objects.
save	SAVE a timeformat object to file.
set	SET set timeformat properties.
string	STRING writes a command string that can be used to recreate the input timeformat object.
subsasgn	SUBSASGN define index assignment for timeformat properties.
subsref	SUBSREF Define field name indexing for timeformat objects.
timeformat	TIMEFORMAT timeformat object class constructor.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@timeformat/private >

Index for classes/@timeformat/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: timeformat/private %%%%%%%% %%%%%%%%%%%%%%
retrieve	RETRIEVE timeformat objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 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.
display	DISPLAY overloads display functionality for timespan objects.
eq	EQ overloads the == operator for timespan objects.
get	GET get timespan properties.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given timespan has all the correct fields of the correct type.
ne	NE overloads the ~= operator for timespan objects.
save	SAVE a timespan object to file.
set	SET set timespan properties.
string	STRING writes a command string that can be used to recreate the input timespan object.
subsasgn	SUBSASGN define index assignment for timespan properties.
subsref	SUBSREF Define field name indexing for timespan objects.
timespan	TIMESPAN timespan object class constructor.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@timespan/private >

Index for classes/@timespan/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: timespan/private %%%%%%%%%%%%%%%%%%%%%%
compute_interval	COMPUTE_INTERVAL compute the interval of the time span.
retrieve	RETRIEVE timespan objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@tsdata >

Index for classes/@tsdata

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: tsdata %%%%%%%%%%%%%%%%%%%%%%
char	CHAR convert a param object into a string.
display	DISPLAY overloads display functionality for tsdata objects.
eq	EQ overloads the == operator for tsdata objects.
get	GET get tsdata properties.
get_xy_values	GET_XY_VALUES returns the x-axis and/or y-axis values.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given tsdata has all the correct fields of the correct type.
ne	NE overloads the ~= operator for tsdata objects.
reshapeT	RESHAPET reshape the time-vector to match the x vector.
save	SAVE a tsdata object to file.
set	SET sets a tsdata property.
setTime	SETTIME sets the time-vector of a tsdata object based on the length of the x vector and the sample rate.
set_xy_axis	SET_XY_AXIS set the x-axis and y-axis values.
single_operation	SINGLE_OPERATION implements specified operator overload for tsdata objects.
subsasgn	SUBSASGN define index assignment for for tsdata properties.
subsref	SUBSREF Define field name indexing for tsdata objects.
tsdata	TSDATA time-series object class constructor.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@tsdata/private >

Index for classes/@tsdata/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%%%% class: tsdata/private %%%%%%%%%%%%%%%%%%%%%%%
retrieve	RETRIEVE tsdata objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@xydata >

Index for classes/@xydata

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: xydata %%%%%%%%%%%%%%%%%%%%%%
char	CHAR convert a param object into a string.
display	DISPLAY overloads display functionality for xydata objects.
eq	EQ overloads the == operator for xydata objects.
get	GET gets xydata properties.
get_xy_values	GET_XY_VALUES returns the x-axis and/or y-axis values.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given xydata has all the correct fields of the correct type.
ne	NE overloads the ~= operator for xydata objects.
reshapeX	RESHAPEX reshape the X to match the Y vector.
save	SAVE an xydata object to file.
set	SET sets a xydata property.
set_xy_axis	SET_XY_AXIS set the x-axis and y-axis values.
single_operation	SINGLE_OPERATION implements specified operator overload for xydata objects.
subsasgn	SUBSASGN define index assignment for xydata properties.
subsref	SUBSREF Define field name indexing for xydata objects.
xydata	XYDATA X-Y data object class constructor.

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@xydata/private >

Index for classes/@xydata/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%%%% class: xydata/private %%%%%%%%%%%%%%%%%%%%%%%%
retrieve	RETRIEVE xydata objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@xyzdata >

Index for classes/@xyzdata

Matlab files in this directory:

display	DISPLAY overloads display functionality for xyzdata objects.
eq	EQ overloads the == operator for xyzdata objects.
get	GET gets xyzdata properties.
get_xyz_values	GET_XYZ_VALUES returns the x-axis and/or y-axis and/or z-axis values.
isfield	ISFIELD tests if the given field is one of the object properties.
isvalid	ISVALID tests if the given xyzdata has all the correct fields of the correct type.
ne	NE overloads the ~= operator for xyzdata objects.
save	SAVE an xyzdata object to file.
set	SET sets an xyzdata property.
single_operation	SINGLE_OPERATION implements specified operator overload for xyzdata objects.
subsasgn	SUBSASGN define index assignment for xyzdata properties.
subsref	SUBSREF Define field name indexing for xyzdata objects.
xyzdata	XZYDATA X-Y-Z data object class constructor.

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for classes/@zero/private >

Index for classes/@zero/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% class: zero/private %%%%%%%%%%%% %%%%%%%%%%
pfq2ri	Convert frequency/Q pole/zero representation into real
pri2fq	Convert comlpex pole/zero into frequency/Q pole/zero representation.
retrieve	RETRIEVE zero objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 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 overloads the foo function for Analysis Objects.
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 Tue 26-Feb-2008 10:52:41 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 %%%%%%%%%%%%%%%%%%%%%%
explore_ao	EXPLORE_AO 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 Tue 26-Feb-2008 10:52:41 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 %%%%%%%%%%%%%%%%%%%%%%
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 Tue 26-Feb-2008 10:52:41 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_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_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 Tue 26-Feb-2008 10:52:41 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 Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index

Index for m/helper >

Index for m/helper

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% path: m/helper %%%%%%%%%%%%%%%%%%%%%%
collect_inputs	COLLECT_INPUTS Collect the AOs and Plists.
gcbsh	GCBSH gets the handles for the currently selected blocks.
get_curr_m_file_path	GET_CURR_M_FILE_PATH returns the path for a mfile.
ltpda_classes	LTPDA_CLASSES lists all the LTPDA object types.
ltpda_deg2rad	LTPDA_DEG2RAD Convert degrees to radians
ltpda_disp	LTPDA_DISP display a formatted string to screen.
ltpda_fields2list	LTPDA_FIELDS2LIST splits a string containing fields seperated by ','
ltpda_filescan	LTPDA_FILESCAN recursively scans the given directory for files that end in 'ext'
ltpda_fq2ri	LTPDA_FQ2RI Convert frequency/Q pole/zero representation into real
ltpda_fq2ri2	LTPDA_FQ2RI2 Convert frequency/Q pole/zero representation into real
ltpda_funchash	LTPDA_FUNCHASH compute MD5 hash of an LTPDA function.
ltpda_getmax	LTPDA_GETMAX get the index and value of the maximum value in an array.
ltpda_hash	LTPDA_HASH - Convert an input variable into a message digest using any of
ltpda_io_collect	LTPDA_IO_COLLECT collects all AOs and plists from the inputs.
ltpda_isobject	LTPDA_ISOBJECT checks that the input objects are one of the LTPDA object
ltpda_isuserobject	LTPDA_ISUSEROBJECT checks that the input objects are one of the LTPDA user object
ltpda_label	LTPDA_LABEL makes the input string into a suitable string for using on plots.

ltpda_mat2str	LTPDA_MAT2STR overloads the mat2str operator to set the precision at a central place.
ltpda_matchvectors	LTPDA_RESAMPLE make the two vectors have the same length.
ltpda_num2str	LTPDA_NUM2STR uses sprintf to convert a data vector to a string with a fixed precision.
ltpda_phase	LTPDA_PHASE return the phase in degrees for a given complex input.
ltpda_rand	LTPDA_RAND return a random number between r1 and r2
ltpda_ri2fq	LTPDA_RI2FQ Convert complex pole/zero into frequency/Q pole/zero representation.
ltpda_saveobj	SAVEOBJ saves an object to a file.
ltpda_str2cells	LTPDA_STR2CELLS Take a single string and separate out individual "elements" into a new cell array.
ltpda_strs2cells	LTPDA_STRS2CELLS convert a set of input strings to a cell array.
ltpda_structcat	LTPDA_STRUCTCAT concatonate structures to make one large structure.
ltpda_structidx	LTPDA_STRUCTIDX Index a structure.
ltpda_unwrapdeg	LTPDA_UNWRAPDEG Unwrap a phase vector given in degrees.
ltpda_user_classes	ltpda_user_classes lists all the LTPDA user object types.
ltpda_xml_read_datasamples	LTPDA_XML_READ_DATASAMPLES read data samples from the given node of an XML file.
ltpda_xmlread	LTPDA_XMLREAD Reads a XML object
ltpda_xmlwrite	LTPDA_XMLWRITE Add an object to a xml DOM project.
phase	PHASE return the phase in degrees for a given complex input.
rstruct	RSTRUCT recursively converts an object into a structure. This is the same
strpad	STRPAD Pads a string with blank spaces until it is N characters long.
wrapstring	WRAPSTRING wraps a string to a cell array of strings with each cell less than n characters long.

Subsequent directories:

- CVS
- generic_class_fcn

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for m/helper/generic_class_fcn >

Index for m/helper/generic_class_fcn

Matlab files in this directory:

generic_eq	EQ overloads the == operator for general objects.
generic_get	GET get a property of a general object.
generic_set	SET set a property of a general object.

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 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 Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index

Index for m/mysql >

Index for m/mysql

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% path: m/mysql %%%%%%%%%%%%%%%%%%%%%%
logindlg	LOGINDLG Dialog for visually secure login.
ltpda_dbquery	LTPDA_DBQUERY query an AO repository database.
ltpda_getAOsInTimeSpan	LTPDA_GETAOSINTIMESPAN performs high-level queries to retrieve AOs from an LTPDA repository.
ltpda_insert	LTPDA_INSERT inserts values into a single row of a table using JDBC driver
ltpda_obj_retrieve	LTPDA_OBJ_RETRIEVE retrieves a collection of objects from an LTPDA repository.
ltpda_obj_submit	LTPDA_OBJ_SUBMIT submits the given collection of objects to an LTPDA Repository.
mysql_connect	MYSQL_CONNECT connects to an AO repository and returns the connection object.
mysql_getMD5hash	MYSQL_GETMD5HASH gets the md5 hash string of a given object ID.
mysql_getMaxId	MYSQL_GETMAXID get the maximum Id from the objs table.
mysql_getObjIds	MYSQL_GETOBJIDS gets a list of object ids from a given collection id.
mysql_getObjType	MYSQL_GETOBJTYPE gets the object type associated with the given object id.
mysql_getUserID	MYSQL_GETUSERID gets the user ID number corresponding to the given user name.
mysql_getXdoc	MYSQL_GETXDOC retrieves an object with given id from the LTPDA

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for m/noisegenerator >

Index for m/noisegenerator

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% path: m/noisegenerator %%%%%% %%%%%%%%%%%%%%%%
ltpda_noisegen	LTPDA_NOISEGEN uses pzm2ab and franklin to generate a time-series
ngconv	NGINIT 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

Subsequent directories:

- CVS
- private

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for m/noisegenerator/private >

Index for m/noisegenerator/private

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%%%% path: m/noisegenerator/private %%%%%%%%%%%%%%%%%%%%%%%%
conv_noisegen	CONV_NOISEGEN calls the matlab function conv.m to convolute poles and
fq2fac	FQ2FAC is a private function and is called by ngconv.m which can be found in the

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for m/plottools >

Index for m/plottools

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% path: m/plottools %%%%%%%%%%%%%%%%%%%%%%
islinespec	ISLINESPEC checks a string to the line spec syntax.
ltpda_allgrid	LTPDA_ALLGRID Set all the grids to 'state'
ltpda_allxaxis	LTPDA_ALLXSCALE Set all the x scales on the current figure.
ltpda_allxlabel	LTPDA_ALLXLABEL Set all the x-axis labels on the current figure.
ltpda_allxscale	LTPDA_ALLXSCALE Set all the x scales on the current figure.
ltpda_allyaxis	LTPDA_ALLYAXIS Set all the yaxis ranges on the current figure.
ltpda_allylabel	LTPDA_ALLYLABEL Set all the y-axis labels on the current figure.
ltpda_allyscale	LTPDA_ALLYSCALE Set all the Y scales on the current figure.
ltpda_cscale	LTPDA_CSCALE Set the color range of the current figure
ltpda_legendAdd	LTPDA_LEGENDADD Add a string to the current legend.
ltpda_msuptitle	LTPDA_MSUPTITLE Puts a title above all subplots.
ltpda_xaxis	LTPDA_XAXIS Set the X axis range of the current figure
ltpda_xscale	LTPDA_XSCALE Set the X scale of the current axis
ltpda_xticks	LTPDA_XTICKS set the input vector as the y-ticks of the current axis.
ltpda_yaxis	LTPDA_YAXIS Set the Y axis range of the current figure
ltpda_yscale	LTPDA_YSCALE Set the Y scale of the current axis
ltpda_yticks	LTPDA_YTICKS set the input vector as the y-ticks of the current axis.
ltpda_zaxis	LTPDA_ZAXIS Set the Z axis range of the current figure
ltpda_zscale	LTPDA_ZSCALE Set the Z scale of the current axis

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 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_arma_freq	LTPDA_ARMA_FREQ estimates the ARMA parameters of the transfer function
ltpda_cohere	LTPDA_COHERE makes coherence estimates of the time-series objects
ltpda_compute_f	LTPDA_COMPUTE_F compute the frequency and resolution vectors for use with mlpsd.
ltpda_cpsd	LTPDA_CPSD makes cross-spectral density estimates of the time-series objects.
ltpda_lcohere	LTPDA_LCOHERE implement weighted coherence estimation computed on a log frequency axis.
ltpda_lcpsd	LTPDA_LCPSD implement weighted cross-power estimation computed on a log frequency axis.
ltpda_linedetect	LTPDA_LINEDETECT find spectral lines in the ao/fsdata objects.
ltpda_lpsd	LTPDA_LPSD implement LPSD algorithm for analysis objects.
ltpda_ltf_plan	LTPDA_LTF_PLAN computes all input values needed for the LPSD and LTFE
ltpda_ltfe	LTPDA_LTFE implement transfer-function estimation computed on a log frequency axis.
ltpda_lxspec	LTPDA_LXSPEC performs cross-spectral analysis of various forms.
ltpda_nfest	LTPDA_NFEST Estimate the noise floor of the given frequency-series AO.
ltpda_psd	LTPDA_PSD a psd estimation based on pwelch from MATLAB.
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.
ltpda_xspec	LTPDA_XSPEC performs cross-spectral analysis of various forms.

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 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_arma_time	LTPDA_ARMA_TIME estimates the ARMA parameters of the transfer function
ltpda_chkts	LTPDA_CHKTS checks that the time-series in the input AO(s) are evenly sampled according to the sample rate and an allowed error.
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_timedomainfit	LTPDA_TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to a target time-series AO.
ltpda_xcorr	LTPDA_XCORR makes cross-correlation estimates of the time-series
polydetrend	POLYDETREND detrends the input data vector with a polynomial.

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for m/timetools >

Index for m/timetools

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% path: m/timetools %%%%%%%%%%%%%%%%%%%%%%
ltpda_gps2utc	LTPDA_GPS2UTC converts GPS seconds to UTC time.
ltpda_reformat_date	LTPDA_REFORMAT_DATE reformats the input date
ltpda_utc2gps	LTPDA_UTC2GPS Converts UTC time to GPS seconds.

Subsequent directories:

- CVS
- statespacefunctions

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for m/timetools/statespacefunctions >

Index for m/timetools/statespacefunctions

Matlab files in this directory:

ltpda_jacobian	subfunction to handle exceptions when calling the jacobian function
ltpda_kalman_filter	Kalman filter
ltpda_simulate_discrete	simulates a discrete state space system
ltpda_ss_assemble	assembles embedded subsytems, with exogenous inputs
ltpda_ss_assemble_arrange	makes meta-system representing assembled embedded subsytems with cells
ltpda_ss_assemble_fusion	SystemFlatten makes a system out of assembled subsystems
ltpda_ss_check	ltpda_ss_check throws an error is something is wrong is found in the subsystem.
ltpda_ss_iir2ss	ltpda_miir2ss converts a iir filter model into a subsystem.
ltpda_ss_kalman_filter	Kalman filter
ltpda_ss_make	ltpda_ss_make is the generic function to decalre or load a subsystem.
ltpda_ss_modify	paramSubstitute enables to modify parameters and retrieve numerical
ltpda_ss_modify_time_setup	timeModify is the generic function to decalre or load a subsystem.
ltpda_ss_pz2ss	% conversion from a pole zero model
ltpda_ss_ss2iir	makeSubsystem is the generic function to decalre or load a subsystem.
utp_ltpda_simulate_discrete	tests ltpda_simulate_discrete which simulates a discrete state space system
utp_ltpda_ss_assemble_1	test function for utp_ltpda_ss_assemble using 3 subsystems, random dims.
utp_ltpda_ss_assemble_2	test function for ltpda_ss_assemble - syntaxic
utp_ltpda_ss_iir2ss	test function for ltpda_ss_miir2ss

utp_ltpda_ss_kalman_filter	tests ltpda_simulate_discrete which simulates a discrete state space system
utp_ltpda_ss_make	tests utp_ltpda_ss_make
utp_ltpda_ss_modify	tests ltpda_ss_modify
utp_ltpda_ss_modify_time_setup	tests ltpda_ss_modify_time_setup with both syntaxic and numerical test.
utp_ltpda_ss_pz2ss	test conversion from state-space to pole zero model
utp_ltpda_ss_random_generate	utp_ltpda_ss_random_generate subfunction generates a random system
utp_ltpda_ss_ss2iir	make is the generic function to decalre or load a subsystem.

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

< Master index Index for mex >

Index for mex

Matlab files in this directory:

ltpda_dft LTPDA_DFT computes the DFT of a signal at one frequency. MEX
ltpda_polyreg LTPDA_POLYREG detrends an input vector with a given order. MEX

Subsequent directories:

- CVS

Generated on Tue 26-Feb-2008 10:52:41 by **m2html** © 2003

ltpda_saveobj

PURPOSE

SAVEOBJ saves an object to a file.

SYNOPSIS

function ltpda_saveobj(a, pl)

DESCRIPTION

```
 SAVEOBJ saves an object to a file.

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

 DESCRIPTION: SAVEOBJ saves an object to a file.

 CALL:      saveobj(obj, pl)

 INPUTS:     obj  - an object (for example, an AO)
             pl   - parameter list with a 'filename' parameter

 OUTPUTS:

 VERSION: $Id: ltpda_saveobj.m,v 1.2 2008/02/13 10:13:54 ingo Exp $

 HISTORY: 28-08-2007 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION

This function calls:

- ltpda_xmlwrite LTPDA_XMLWRITE Add an object to a xml DOM project.

This function is called by:

SOURCE CODE

```
function ltpda_saveobj(a, pl)
% SAVEOBJ saves an object to a file.
```

```
%
%%%
```

save

PURPOSE ^

SAVE a pzmodel object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE a pzmodel object to file.

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

 DESCRIPTION: SAVE overloads the save method for pzmodel objects.

 CALL:        Save a pzmodel object as a .mat file.
                >> save(pzm, 'blah.mat')
              Save a pzmodel object as an XML file.
                >> save(pzm, 'blah.xml')
              Save a pzmodel object to file specified in plist.
                >> save(pzm, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.4 2008/02/15 16:27:16 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(pzmodel, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(pzmodel, 'Version')

 The following call returns a string that contains the routine category:

 >> category = save(pzmodel, 'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pzmodel object into a string.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE a pzmodel object to file.
%
%%%
```

```

ALGONAME = mfilename;
VERSION  = '$Id: save.m,v 1.4 2008/02/15 16:27:16 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'pzmodel') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: ao/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%   class: ao/private   %%%%%%%%%%%%%%%%%%

   <a href="matlab:help ao/private/aooperate">ao/private/aooperate</a>            -
implements specified operator overload for AOs.
   <a href="matlab:help ao/private/fromxml">ao/private/fromxml</a>              -
read from part of an XML tree and create an analysis object.
   <a href="matlab:help ao/private/retrieve">ao/private/retrieve</a>             -
analysis objects with given ids from the LTPDA repository specified by the input
   <a href="matlab:help ao/private/xdoc2ao">ao/private/xdoc2ao</a>              -
converts a DOM object to an AO.
   <a href="matlab:help ao/private/xml_add_data">ao/private/xml_add_data</a>
-  add a data object to the given node of the input XML DOM
   <a href="matlab:help ao/private/xml_add_hist">ao/private/xml_add_hist</a>
-  add an history object to the given node of the input XML DOM
   <a href="matlab:help ao/private/xml_add_miir">ao/private/xml_add_miir</a>
-  add an MIIR object to the given node of the input XML DOM
   <a href="matlab:help ao/private/xml_add_param">ao/private/xml_add_param</
a>        -  add a parameter object to the given node of the input XML
   <a href="matlab:help ao/private/xml_add_plist">ao/private/xml_add_plist</
a>        -  add a plist object to the given node of the input XML DOM
   <a href="matlab:help ao/private/xml_add_pole">ao/private/xml_add_pole</a>
-  add a pole object to the given node of the input XML DOM
   <a href="matlab:help ao/private/xml_add_provenance">ao/private/
xml_add_provenance</a>   -  add a provenance object to the given node of the input
XML DOM
   <a href="matlab:help ao/private/xml_add_pzmodel">ao/private/xml_add_pzmodel</
a>      -  add a pzmodel object to the given node of the input XML DOM
   <a href="matlab:help ao/private/xml_add_specwin">ao/private/xml_add_specwin</
a>      -  adds specwin object to the given node of the xml input
   <a href="matlab:help ao/private/xml_add_time">ao/private/xml_add_time</a>
-  add a time object to the given node of the input XML DOM
   <a href="matlab:help ao/private/xml_add_timeformat">ao/private/
xml_add_timeformat</a>   -  add a time object to the given node of the input XML DOM
   <a href="matlab:help ao/private/xml_add_zero">ao/private/xml_add_zero</a>
-  add a zero object to the given node of the input XML DOM
```

```
   <a href="matlab:help ao/private/xml_read">ao/private/xml_read</a>              -
reads an xml file and returns an analysis object.
   <a href="matlab:help ao/private/xml_read_data">ao/private/xml_read_data</
a>        -  read a data object from the given node of an XML DOM
   <a href="matlab:help ao/private/xml_read_datasamples">ao/private/
xml_read_datasamples</a> -  read data samples from the given node of an XML file.
   <a href="matlab:help ao/private/xml_read_hist">ao/private/xml_read_hist</
a>        -  read history object from the given node of an XML DOM
   <a href="matlab:help ao/private/xml_read_miir">ao/private/xml_read_miir</
a>        -  read an MIIR object from the given input node.
   <a href="matlab:help ao/private/xml_read_param">ao/private/xml_read_param</
a>       -  read a parameter from the given input node.
   <a href="matlab:help ao/private/xml_read_plist">ao/private/xml_read_plist</
a>       -  read a plist object from the given input node.
   <a href="matlab:help ao/private/xml_read_pole">ao/private/xml_read_pole</
a>        -  read a pole object from the given input node.
   <a href="matlab:help ao/private/xml_read_provenance">ao/private/
xml_read_provenance</a>  -  read provenance object from the given node of an XML DOM
   <a href="matlab:help ao/private/xml_read_pzmodel">ao/private/xml_read_pzmodel</
a>     -  read a pzmodel object from the given input node.
   <a href="matlab:help ao/private/xml_read_specwin">ao/private/xml_read_specwin</
a>     -  read a specwin object from the given input node.
   <a href="matlab:help ao/private/xml_read_time">ao/private/xml_read_time</
a>        -  read time object from the given node of an XML DOM
   <a href="matlab:help ao/private/xml_read_timeformat">ao/private/
xml_read_timeformat</a>  -  read time object from the given node of an XML DOM
   <a href="matlab:help ao/private/xml_read_zero">ao/private/xml_read_zero</
a>        -  read a zero object from the given input node.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

format

PURPOSE ^

FORMAT Returns the time in the handover format.

SYNOPSIS ^

function string = format(tt, format_str)

DESCRIPTION ^

```
 FORMAT Returns the time in the handover format.

%%%
```

```
%
%%%
```

ltpda_simulate_discrete

PURPOSE ^

simulates a discrete state space system

SYNOPSIS ^

function varargout = ltpda_simulate_discrete(varargin)

DESCRIPTION ^

```
simulates a discrete state space system
%%%
```

This function is called by:

- utp_ltpda_simulate_discrete tests ltpda_simulate_discrete which simulates a discrete state space system
- utp_ltpda_ss_kalman_filter tests ltpda_simulate_discrete which simulates a discrete state space system

SOURCE CODE ^

```
function varargout = ltpda_simulate_discrete(varargin)
%simulates a discrete state space system
%%%
```

```
    return;
elseif isequal(varargin{1}, 'Category')
    varargout = CATEGORY;
    return;
end
syst = varargin{2};
ao_in_list = varargin{1};

%% retrieve system infos
A = find(syst, 'AMAT');
B = find(syst, 'BMATS');
C = find(syst, 'CMAT');
D = find(syst, 'DMATS');
A = A{1};
C = C{1};
nss = size(A,1);
%check iniside A, Bs, c, Ds there is no symbolic variable
Xini = find(syst, 'XINI');
timestep = find(syst, 'TIMESTEP');
NBINPUTS = find(syst, 'NBINPUTS');
INPUTISUSED = find(syst, 'INPUTISUSED');
INPUTNAMES = find(syst, 'INPUTNAMES');
INPUTSIZES = find(syst, 'INPUTSIZES');
XISOUTPUT = find(syst, 'XISOUTPUT');
YISOUTPUT = find(syst, 'YISOUTPUT');



%% read out AO (time-series data)
ao_in = cell(NBINPUTS,1);
for i=1:NBINPUTS
    if INPUTISUSED(i)
        ao_in{i} = find(ao_in_list, INPUTNAMES{i});
        if isequal(class(ao_in{i}), 'ao')
            if not(isequal(ao_in{i}.data.x,{})) %making sure ao is not empty
                t = ao_in{i}.data.x;
                dt = diff(t);
                if sum(not(isequal(dt,timestep)))>1
                    msg = ['error in input number ', i , ' called ', INPUTNAMES{i}, 'because wrong timestep for simulation'];
                    error(msg);
                end
                if not( isequal( size(B{i},1), size(D{i},1), size(ao_in{i}.data.y,1) ) )
                    msg = ['error in input number ', i , ' called ', INPUTNAMES{i}, 'because wrong input size'];
                    error(msg);
                end
            end
        end
    end
end
%% getting correct number of samples
```

```
nsamples = inf;
for i=1:NBINPUTS
    if isequal(class(ao_in{i}), 'ao')
        nsamples = min(nsamples,size(ao_in{i}.data.y,1));
    end
end
if nsamples == inf % case there is no input!
    nsamples = 0;
end

%% generating random data
for i=1:NBINPUTS
    if isequal(class(ao_in{i}), 'plist')
        sigma = find(ao_in{i}, 'SIGMA');
        mean = find(ao_in{i}, 'MEAN');
        if sigma==0
            ao_in{i} = ao(ones(nsamples,INPUTSIZES(i))*diag(mean));
        else
            ao_in{i} = ao(ones(nsamples,INPUTSIZES(i))*diag(mean) + randn(nsamples,INPUTSIZES(i))*sigma);
        end
    end
end

%% simulation first step
x = zeros(nss, nsamples);
y = zeros(size(C,1), nsamples);
x(:,1) = Xini;
IncrementY = zeros(size(C,1),1);
for i=1:NBINPUTS
    if INPUTISUSED(i)
        IncrementY = IncrementY +  D{i}* ao_in{i}.data.y(1,:)' ;
    end
end
y(:,1) = C* x(:,1) +IncrementY;

%% simulation loop
for k = 2:nsamples
    IncrementX = zeros(nss,1);
    IncrementY = zeros(size(C,1),1);
    for i=1:NBINPUTS
        if INPUTISUSED(i)
            IncrementX = IncrementX +  B{i}*ao_in{i}.data.y(k,:)' ;
            IncrementY = IncrementY +  D{i}*ao_in{i}.data.y(k,:)' ;
        end
    end
    x(:,k) = A* x(:,k-1) + IncrementX ;
    y(:,k) = C* x(:,k) + IncrementY ;
end

%% data object
out_dat = zeros(0, nsamples);
if XISOUTPUT
```

```
    out_dat = [out_dat; x];
end
if YISOUTPUT
    out_dat = [out_dat; y];
end

if exist('t','var')
    ts = tsdata(t,out_dat) ;
else
    ts = tsdata(1:nsamples,out_dat) ;
end

%% history object, old history from ao_in plus new
% hi_old = ao_in.hist ;
hi_new = history('simulate_discrete','version', syst);
% hi = [hi_old hi_new];
% h = history(hi);
h = history(hi_new);

%% construct output analysis object
varargout  = ao(ts,h) ;
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

save

PURPOSE ^

SAVE a specwin object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE a specwin object to file.

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

 DESCRIPTION: SAVE overloads the save method for specwin objects.

 CALL:        Save a specwin object as a .mat file.
                >> save(w, 'blah.mat')
              Save a specwin object as an XML file.
                >> save(w, 'blah.xml')
              Save a specwin object to file specified in plist.
                >> save(w, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.3 2008/02/13 13:46:07 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(specwin, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(specwin, 'Version')

 The following call returns a string that contains the routine category:

 >> category = save(specwin, 'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a specwin object into a string.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE a specwin object to file.
%
%%%
```

```

VERSION  = '$Id: save.m,v 1.3 2008/02/13 13:46:07 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'specwin') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: cdata %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: cdata   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help cdata/cdata">cdata/cdata</a>            -  constant data
object class constructor.
   <a href="matlab:help cdata/char">cdata/char</a>             -  convert a cdata
object into a string.
   <a href="matlab:help cdata/display">cdata/display</a>          -  implement
terminal display for cdata object.
   <a href="matlab:help cdata/eq">cdata/eq</a>               -  overloads the ==
operator for cdata objects.
   <a href="matlab:help cdata/get">cdata/get</a>              -  get csdata
properties.
   <a href="matlab:help cdata/get_xy_values">cdata/get_xy_values</a>    -  returns
the values in vals.
   <a href="matlab:help cdata/isfield">cdata/isfield</a>          -  tests if the
given field is one of the object properties.
   <a href="matlab:help cdata/ne">cdata/ne</a>               -  overloads the ~=
operator for cdata objects.
   <a href="matlab:help cdata/save">cdata/save</a>             -  a cdata object to
file.
   <a href="matlab:help cdata/set">cdata/set</a>              -  set a cdata
property.
   <a href="matlab:help cdata/set_xy_axis">cdata/set_xy_axis</a>      -  set the x-
axis and y-axis values.
   <a href="matlab:help cdata/single_operation">cdata/single_operation</a> -
implements specified operator overload for cdata objects.
   <a href="matlab:help cdata/subsasgn">cdata/subsasgn</a>         -  define index
assignment for vectors in tsdata.
   <a href="matlab:help cdata/subsref">cdata/subsref</a>          -  Define field
name indexing for cdata objects.
   <a href="matlab:help cdata/xml">cdata/xml</a>              -  convert a cdata
object to an DOM XML object.
   <a href="matlab:help cdata/xmladd">cdata/xmladd</a>           -  add a cdata
object to the given node of the input XML DOM
```

```
   <a href="matlab:help cdata/xmlparse">cdata/xmlparse</a>         -  creates a
cdata from a cdata xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 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 1.6 2008/02/22 11:12:46 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function polzero = fq2fac(f,q)
% 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 1.6 2008/02/22 11:12:46 hewitson Exp $
%

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
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/01/29 10:50:26 josep Exp $

 HISTORY:     24-12-2007 J Sanjuan
                Creation
```

```
%%%
```

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

```

    % 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(pl, 'kspike'); % kspike*sigma definition
    method = find(pl, 'method'); % method of spike-values substitution
    pl = pset(pl, param('gain', 1)); % gain of the filter
    pl = pset(pl, param('type', 'highpass')); % type of the filter
    fs = plist();
    fs = append(fs, param('fs', d.fs));
    pl = combine(pl, fs); % determination of the sampling frequency of the input AO

% High-pass filtering data
xfiltered = filtfilt(a, miir(pl));

% 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 = set(ts, 'name', sprintf('cleaned data of %s', d.name));
ts = set(ts, 'yunits', d.yunits);
```

```
ts = set(ts, 'xunits', d.xunits);

% create new output history
h = history(ALGONAME, VERSION, pl, a.hist);
h = set(h, 'invars', invars);

% make output analysis object
b = ao(ts, h);

% set name
if isempty(invars{i})
    n = a.name;
else
    n = invars{i};
end

b = setnh(b, 'name', sprintf('spikecleaning(%s)', n));

% add to output array
bo = [bo b];

varargout{1} = bo;
end

%-------------------------------
% Get default parameters
function plo = getDefaultPL();

disp('* creating default plist...');
plo = plist();
plo = append(plo, param('kspike', 3.3));
plo = append(plo, param('fc', 0.025));
plo = append(plo, param('order', 2));
plo = append(plo, param('ripple', 0.5));
plo = append(plo, param('method', 'random'));
disp('* done.');
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

save

PURPOSE ^

SAVE a time object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE a time object to file.

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

 DESCRIPTION: SAVE overloads the save method for time objects.

 CALL:        Save a time object as a .mat file.
                >> save(t, 'blah.mat')
              Save a time object as an XML file.
                >> save(t, 'blah.xml')
              Save a time object to file specified in plist.
                >> save(t, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.3 2008/02/13 16:31:26 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(time, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(time,'Version')

 The following call returns a string that contains the routine category:

 >> category = save(time,'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a time object into a string.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE a time object to file.
%
%%%
```

```

VERSION  = '$Id: save.m,v 1.3 2008/02/13 16:31:26 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'time') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: cdata/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: cdata/private   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help cdata/private/fromxml">cdata/private/fromxml</a>  -  read
from part of an XML tree and create a cdata object.
   <a href="matlab:help cdata/private/retrieve">cdata/private/retrieve</a> -  cdata
objects with given ids from the LTPDA repository specified by the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

fsdata

PURPOSE ^

FSDATA frequency-series object class constructor.

SYNOPSIS ^

function fsd = fsdata(varargin)

DESCRIPTION ^

```
 FSDATA frequency-series object class constructor.

%%%
```

```
 HISTORY: 30-01-2007 Hewitson
             Creation

%%%
```

```
%                                (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.
%
% HISTORY: 30-01-2007 Hewitson
%             Creation
%
%%%
```

```
if nargin == 0

  fsd = init();

elseif nargin == 1
  %%%%%%%%%%% From File %%%%%%%%%%%%%%%%
  if ischar(varargin{1})

    filename = varargin{1};
    [path, name, ext, vers] = fileparts(filename);
    switch ext
      case '.mat'
        fsd = load(filename);
        fsd = fsd.a;
      case '.xml'
        root_node = xmlread(filename);
        fsd = ltpda_xmlread(root_node, 'fsdata');
      otherwise
        error('### Unknown file type.');
    end
  %%%%%%%%%%  fsd = fsdata(fsdata)   %%%%%%%%%%
  elseif isa(varargin{1}, 'fsdata')

    fsd = varargin{1};

  %%%%%%%%%%  fsd = fsdata(struct)   %%%%%%%%%%
  elseif isstruct(varargin{1})

    fsd = init();

    fields = fieldnames(varargin{1});
    for ii = 1:length(fields)
      field = fields{ii};

      %%% created -> time-object
      if strcmp(field, 'created')
        created       = varargin{1}.created;
        if isstruct(created)
          created = time(created);
        end
        fsd.created     = created;
      %%% All other
      else
        try
          fsd.(field) = varargin{1}.(field);
        catch
          error('### The field ''%s'' in the struct is not a fsdata property.', field)
        end
      end
    end

  %%%%%%%%%%  fsd = fsdata(y_vector)   %%%%%%%%%%
```

```
  else
    fsd    = init();
    fsd.fs = 1;
    fsd.y  = varargin{1};

    fsd    = setFreq(fsd);
  end

  % Unify the y-axis and y-axis
  if size(fsd.y,2) > size(fsd.y,1)
    fsd.y = [fsd.y].';
  end
  if size(fsd.x, 1) ~= size(fsd.y,1)
    fsd.x = [fsd.x].';
  end

%%%%%%%%%%  fsd = fsdata(x_vector, y_vector)   %%%%%%%%%%
elseif nargin == 2
  %%%%%%%%%%% From DATABASE
  if isa(varargin{1}, 'database')
    fsd = retrieve(varargin{1}, varargin{2:end});
  elseif length(varargin{1}) == length(varargin{2})

    fsd    = init();
    fsd.x  = varargin{1};
    fsd.y = varargin{2};
    fsd.fs = fsd.x(end)*2;

    if size(fsd.y,2) > size(fsd.y,1)
      fsd.y = [fsd.y].';
    end
    fsd = reshapeF(fsd);

  %%%%%%%%%%  fsd = fsdata(y_vector, fs)   %%%%%%%%%%
  else
    % check we have fs properly here
    if length(varargin{2}) > 1
      error(['### unknown constructor call. Either x and y should be the' ...
        'same length, or fs should be a single value.']);
    end

    fsd = init();
    fsd.y      = varargin{1};
    fsd.fs      = varargin{2};

    if size(fsd.y,2) > size(fsd.y,1)
      fsd.y = [fsd.y].';
    end
    fsd = setFreq(fsd);
  end


```

```
%%%%%%%%%%  fsd = fsdata(x_vector, y_vector, fs)   %%%%%%%%%%
elseif nargin == 3

  fsd = init();
  fsd.x       = varargin{1};
  fsd.y      = varargin{2};
  fsd.fs      = varargin{3};

  if size(fsd.y,2) > size(fsd.y,1)
    fsd.y = [fsd.y].';
  end
  fsd = reshapeF(fsd);
else
  error('### Unknown number of constructor arguments.');
end

end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_ss_assemble

PURPOSE ^

assembles embedded subsytems, with exogenous inputs

SYNOPSIS ^

function varargout = ltpda_ss_assemble(varargin)

DESCRIPTION ^

```
 assembles embedded subsytems, with exogenous inputs

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

 DESCRIPTION: ltpda_ss_assemble assembles embedded subsytems, with exogenous inputs

 CALL: [Subsys, listInputs] = ltpda_ss_assemble(listSubsystem)

 INPUTS: listSubsystem - parameter list (of parameter lists with subsystem
 format, see ltpda_ss_check for details)

 OUTPUTS: Syst - parameter lists with subsystem format describing the
 global new system

 PARAMETERS: 'NameSubsystem1' - Subsystem1
 'NameSubsystem2' - Subsystem2
 'NameSubsystem3' - Subsystem3
  ...
 ***** There are no default parameter values *****

 See ltpda_ss_check function for information on subsystem plist format
 See ltpad_ss_assemble_arrange function for information on input plist content

 VERSION: $Id: ltpda_ss_assemble.m,v 1.3 2008/02/25 12:43:44 adrien Exp $

 HISTORY: 23-01-2008 A Grynagier
 29-01-2008 A Grynagier

 TO DO : see in 'arrange' and 'fusion' subroutines.  second one needs work
 on D inversion.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

- ltpda_ss_assemble_arrange makes meta-system representing assembled embedded subsytems with cells
- ltpda_ss_assemble_fusion SystemFlatten makes a system out of assembled subsystems

This function is called by:

SOURCE CODE ^

```
function   varargout = ltpda_ss_assemble( varargin )
% assembles embedded subsytems, with exogenous inputs
%
%%%
```

```
if isequal( varargin{1}, 'Version')
    varargout = VERSION;
    return;
elseif isequal(varargin{1}, 'Params')
    varargout = plist();
    return;
elseif isequal(varargin{1}, 'Category')
    varargout = CATEGORY;
    return;
end

listSubsystems = varargin{1};

%% calling subfonctions
% building state space sparce representation with cell arrays
listMeta = ltpda_ss_assemble_arrange(listSubsystems);
% converting into matrices
varargout = ltpda_ss_assemble_fusion(listMeta);

end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

save

PURPOSE ^

SAVE a timeformat object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE a timeformat object to file.

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

 DESCRIPTION: SAVE overloads the save method for timeformat objects.

 CALL:        Save a timeformat object as a .mat file.
                >> save(tf, 'blah.mat')
              Save a timeformat object as an XML file.
                >> save(tf, 'blah.xml')
              Save a timeformat object to file specified in plist.
                >> save(tf, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.3 2008/02/13 16:54:14 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(timeformat, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(timeformat,'Version')

 The following call returns a string that contains the routine category:

 >> category = save(timeformat,'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a timeformat object into a string.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE a timeformat object to file.
%
%%%
```

```
VERSION  = '$Id: save.m,v 1.3 2008/02/13 16:54:14 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'timeformat') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: fsdata %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%   class: fsdata   %%%%%%%%%%%%%%%%%%

   <a href="matlab:help fsdata/char">fsdata/char</a>             -  convert a param
object into a string.
   <a href="matlab:help fsdata/display">fsdata/display</a>          -  implement
terminal display for fsdata object.
   <a href="matlab:help fsdata/eq">fsdata/eq</a>               -  overloads the ==
operator for fsdata objects.
   <a href="matlab:help fsdata/fsdata">fsdata/fsdata</a>           -  frequency-
series object class constructor.
   <a href="matlab:help fsdata/get">fsdata/get</a>              -  get a fsdata
property.
   <a href="matlab:help fsdata/get_xy_values">fsdata/get_xy_values</a>    -  returns
the x-axis and/or y-axis values.
   <a href="matlab:help fsdata/isfield">fsdata/isfield</a>          -  tests if the
given field is one of the object properties.
   <a href="matlab:help fsdata/ne">fsdata/ne</a>               -  overloads the ~=
operator for fsdata objects.
   <a href="matlab:help fsdata/reshapeF">fsdata/reshapeF</a>         -  reshape the
frequency vector to match the xx vector.
   <a href="matlab:help fsdata/save">fsdata/save</a>             -  an fsdata object
to file.
   <a href="matlab:help fsdata/set">fsdata/set</a>              -  set a fsdata
property.
   <a href="matlab:help fsdata/setFreq">fsdata/setFreq</a>          -  sets the
frequency vector of a fsdata object based on the length of the xx vector and the
sample rate.
   <a href="matlab:help fsdata/set_xy_axis">fsdata/set_xy_axis</a>      -  set the x-
axis and y-axis values.
   <a href="matlab:help fsdata/single_operation">fsdata/single_operation</a> -
implements specified operator overload for fsdata objects.
   <a href="matlab:help fsdata/subsasgn">fsdata/subsasgn</a>         -  define index
assignment for fsdata properties.
   <a href="matlab:help fsdata/subsref">fsdata/subsref</a>          -  Define field
name indexing for fsdata objects.
```

```
   <a href="matlab:help fsdata/xml">fsdata/xml</a>              -  convert an fsdata
object to an DOM XML object.
   <a href="matlab:help fsdata/xmladd">fsdata/xmladd</a>          -  add a fsdata
object to the given node of the input XML DOM
   <a href="matlab:help fsdata/xmlparse">fsdata/xmlparse</a>        -  creates a
fsdata from a fsdata xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

gcbsh

PURPOSE ^

GCBSH gets the handles for the currently selected blocks.

SYNOPSIS ^

function varargout = gcbsh()

DESCRIPTION ^

```
 GCBSH gets the handles for the currently selected blocks.

%%%
```

```
% Usage: >> gcbsh
%        >> h = gcbsh
%
% M Hewitson 29-08-07
%
% $Id: gcbsh.m,v 1.3 2008/02/14 18:43:30 hewitson Exp $
%
%%%
```

ltpda_ss_assemble_arrange

PURPOSE ^

makes meta-system representing assembled embedded subsytems with cells

SYNOPSIS ^

function varargout = ltpda_ss_assemble_arrange(varargin)

DESCRIPTION ^

```
 makes meta-system representing assembled embedded subsytems with cells

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

 DESCRIPTION: ltpda_ss_assemble_arrange assembles enbedded subsystems

 CALL: listMeta = ltpda_ss_assemble_arrange(listSubsystems)

 INPUTS: Subsys - parameter list (of parameter lists with subsystem format)

 OUTPUTS: listMeta - parameter list (meta-system format)

 plist('A_META', A_meta, 'B_META', B_meta,  'C_META', C_meta, 'D_META', D_meta,
'BEXT_META', Bext_meta, 'DEXT_META', Dext_meta, ...
     'OY_META', Oy_meta, 'OX_META', Ox_meta,...
     'TIMESTEP', SubsystemTimestep(1), 'XINI_META', SubsystemXini, 'NAMES_META',
SubsystemNames,...
     'PARAMNAMES_META', ParamNames_meta ,'PARAMVALUES_META',
ParamValues_meta ,'PARAMSIGMAS_META', ParamSigmas_meta ,...
     'NBINPUTS_META', NbInputs_meta ,'INPUTNAMES_META',
InputNames_meta ,'INPUTSIZES_META', InputSizes_meta , 'INPUTISUSED_META',
IsUsed_meta);
 where all meta fileds are a cell array of matrices that have matiching
 dimensions or are empty.

 See ltpda_ss_check function for information on subsystem plist format


 VERSION: $Id: ltpda_ss_assemble_arrange.m,v 1.3 2008/02/25 12:43:44 adrien Exp $

 HISTORY: 23-01-2008 A Grynagier

 ***** There are no default parameter values *****
 TO DO : nothing here
```

```
%%%
```

```

%% standard calls for LTPDA function data
display(['starting ' mfilename]);
CATEGORY = 'STATESPACE';
display(['starting ' ALGONAME]);

if isequal( varargin{1}, 'Version')
    varargout = VERSION;
    return;
elseif isequal(varargin{1}, 'Params')
    varargout = plist();
    return;
elseif isequal(varargin{1}, 'Category')
    varargout = CATEGORY;
    return;
end

%% building arrays, collecting data for ease of use
% global data
NSubsystems = length(listSubsystems.params); %number of subsystems
% subsystem data
SubsystemNames = cell(NSubsystems,1);
SubsystemTimestep = zeros(NSubsystems,1);
SubsystemXini = cell(NSubsystems,1);
SubsystemAMat = cell(NSubsystems,1);
SubsystemBMats = cell(NSubsystems,1);
SubsystemCMat = cell(NSubsystems,1);
SubsystemDMats = cell(NSubsystems,1);
% parametric data
SubsystemNbParam = zeros(NSubsystems,1);
SubsystemParamNames = cell(NSubsystems,1);
SubsystemParamValue = cell(NSubsystems,1);
SubsystemParamSigma = cell(NSubsystems,1);
% input data
SubsystemIsUsed = cell(NSubsystems,1);
SubsystemInputNames = cell(NSubsystems,1);
SubsystemInputSizes = cell(NSubsystems,1);
SubsystemNbInputs = zeros(NSubsystems,1);
% output data
XIsOutput = cell(NSubsystems,1);
YIsOutput = cell(NSubsystems,1);
XOutputNames = {};
YOutputNames = {};
for i=1:NSubsystems
    % subsystem data
    SubsystemNames{i} = find(listSubsystems.params(i).val,'NAME');
    SubsystemTimestep(i) = find(listSubsystems.params(i).val,'TIMESTEP');
    SubsystemXini{i} = find(listSubsystems.params(i).val,'XINI');
    SubsystemAMat{i} = find(listSubsystems.params(i).val,'AMAT');
    SubsystemBMats{i} = find(listSubsystems.params(i).val,'BMATS');
    SubsystemCMat{i} = find(listSubsystems.params(i).val,'CMAT');
    SubsystemDMats{i} = find(listSubsystems.params(i).val,'DMATS');
    % parametric data
```

```
    SubsystemParamNames{i} = find(listSubsystems.params(i).val,'PARAMNAMES');
    SubsystemParamValue{i} = find(listSubsystems.params(i).val,'PARAMVALUES');
    SubsystemParamSigma{i} = find(listSubsystems.params(i).val,'PARAMSIGMAS');
    SubsystemNbParam(i) = length(SubsystemParamNames{i});
    % input data
    SubsystemInputNames{i} = find(listSubsystems.params(i).val,'INPUTNAMES');
    SubsystemInputSizes{i} = find(listSubsystems.params(i).val,'INPUTSIZES');
    SubsystemIsUsed{i} = find(listSubsystems.params(i).val,'INPUTISUSED');
    SubsystemNbInputs(i) = length(SubsystemInputNames{i});
    % output data
    XIsOutput{i} = find(listSubsystems.params(i).val,'XISOUTPUT');
    YIsOutput{i} = find(listSubsystems.params(i).val,'YISOUTPUT');
    if XIsOutput{i}==1
        XOutputNames{length(XOutputNames)+1} = SubsystemNames{i}; %#ok<AGROW> %
incrementing the SystemOutput list
    end
    if YIsOutput{i}==1
        YOutputNames{length(YOutputNames)+1} = SubsystemNames{i}; %#ok<AGROW> %
incrementing the SystemOutput list
    end
end
NOutputX = length(XOutputNames); % number of state spaces in global output
NOutputY = length(YOutputNames); % number of outputs as global output

%% Building meta fields for parameters
ParamNames_meta = {};
ParamValues_meta = [];
ParamSigmas_meta = [];
for i=1:NSubsystems %for each subsystem
    for j=1:SubsystemNbParam(i) %for each parameter of a subsystem
        found = 0;
        for k=1:length(ParamNames_meta) %screening through the exsiting array
            if isequal( ParamNames_meta{k}, SubsystemParamNames{i}{j} )
                found = 1;
                % if found, checking values are compatible with redundant
parameter
                if not( ParamValues_meta(k)==SubsystemParamValue{i}(j) &&
ParamSigmas_meta(k)==SubsystemParamSigma{i}(j) )
                    msg = 'error because two parameters used at the same time
have the same name but not the same values';
                    error(msg);
                end
                break;
            end
        end
        if not(found) % if not found, array is incremented
            ParamNames_meta = [ParamNames_meta SubsystemParamNames{i}{j}]; %
#ok<AGROW>
            ParamValues_meta = [ParamValues_meta SubsystemParamValue{i}(j)]; %
#ok<AGROW>
            ParamSigmas_meta = [ParamSigmas_meta SubsystemParamSigma{i}(j)]; %
#ok<AGROW>
        end
```

```
    end
end

%% Building fields for meta-inputs
% meta input list
InputNames_meta = {};
InputSizes_meta = [];
IsUsed_meta = []; % to check that subsystems are taken in account individually when assembled
for i=1:NSubsystems %for each subsystem
    for j=1:SubsystemNbInputs(i) % screeing through all the inputs of the subsystem
        isExternal = 1;
        for k1=1:NSubsystems
            if isequal(SubsystemNames{k1}, SubsystemInputNames{i}{j})
                isExternal = 0; %if it is internal
                break;
            end
        end
        if isExternal
            isListed = 0;
            for k2=1:length(InputNames_meta); % screening through the existing list of inputs
                if isequal( InputNames_meta{k2}, SubsystemInputNames{i}{j} ) % if names match
                    isListed = 1;
                    break;
                end
            end
            if isListed
                if not(InputSizes_meta(k2)==SubsystemInputSizes{i}(j) )
                    msg = 'error because two inputs used at the same time have the same name but not the same size';
                    error(msg);
                end
                IsUsed_meta(k2) = max( SubsystemIsUsed{i}(j), IsUsed_meta(k2) );
            else
                InputNames_meta = [InputNames_meta SubsystemInputNames{i}{j}]; % #ok<AGROW>
                InputSizes_meta = [InputSizes_meta SubsystemInputSizes{i}(j)]; % #ok<AGROW>
                IsUsed_meta = [IsUsed_meta SubsystemIsUsed{i}(j)];        % #ok<AGROW>
            end
        else
            C = SubsystemCMat{k1}{1};
            if not( size(C,1) == SubsystemInputSizes{i}(j) )
                msg = ['error in Subsystem ', SubsystemNames{i} , ', conflicting size of inner input number ', num2str(j) ,' named ', SubsystemInputNames{i}{j} ];
                error(msg);
            end
        end
```

```
    end
end
NbInputs_meta = length(InputNames_meta);

%% Building fields for meta-time model
for i=1:NSubsystems %for each subsystem
    if not( SubsystemTimestep(i)==SubsystemTimestep(1) )
        msg = ['Problem because subsystems have different timestep representation : ', mat2str(SubsystemTimestep)];
        error(msg);
    end
end

%% Building meta-matices
% Declaring meta matrices
A_meta = cell(NSubsystems, NSubsystems);
B_meta = cell(NSubsystems, NSubsystems);
C_meta = cell(NSubsystems, NSubsystems);
D_meta = cell(NSubsystems, NSubsystems);
Bext_meta = cell(NSubsystems,NbInputs_meta);
Dext_meta = cell(NSubsystems,NbInputs_meta);
Ox_meta = cell(NOutputX+NOutputY , NSubsystems);
Oy_meta = cell(NOutputY+NOutputX , NSubsystems);

%fill A and C
for i=1:NSubsystems
    AMAT = find(listSubsystems.params(i).val,'AMAT');
    CMAT = find(listSubsystems.params(i).val,'CMAT');
    A_meta{i,i} = AMAT{1};
    C_meta{i,i} = CMAT{1};
end

%fill B D Bext Dext
for i=1:NSubsystems %going through the subsystems
    for j=1:SubsystemNbInputs(i) %going through the inputs
        for k = 1:NSubsystems %going through the subsystems that might be an input
            if isequal( SubsystemNames{k}, SubsystemInputNames{i}{j} )
                B_meta{i,k} = SubsystemBMats{i}{j};
                D_meta{i,k} = SubsystemDMats{i}{j};
            end
        end
        for k = 1:NbInputs_meta %going through all the exogenous inputs
            if isequal( InputNames_meta{k}, SubsystemInputNames{i}{j} )
                Bext_meta{i,k} = SubsystemBMats{i}{j};
                Dext_meta{i,k} = SubsystemDMats{i}{j};
            end
        end
    end
end

% fill Ox, Oy
for i=1:NSubsystems % going through the subsystems
```

```
    for j=1:NOutputX % screening the State Space
        if isequal(  SubsystemNames{i} , XOutputNames{j} )
            Ox_meta{j,i} = eye( size(SubsystemAMat{i}{1},1) );
        end
    end
    for j=1:NOutputY % screening the local outputs
        if isequal(  SubsystemNames{i} , YOutputNames{j} )
            Oy_meta{j+NOutputX,i} = eye( size(SubsystemCMat{i}{1},1) );
        end
    end
end

%% parsing the meta matrices into a list
varargout = plist('A_META', A_meta, 'B_META', B_meta,  'C_META', C_meta, 'D_META', D_meta, 'BEXT_META', Bext_meta, 'DEXT_META', Dext_meta, ...
    'OY_META', Oy_meta, 'OX_META', Ox_meta,...
    'TIMESTEP', SubsystemTimestep(1), 'XINI_META', SubsystemXini, 'NAMES_META', SubsystemNames,...
    'PARAMNAMES_META', ParamNames_meta ,'PARAMVALUES_META', ParamValues_meta ,'PARAMSIGMAS_META', ParamSigmas_meta ,...
    'NBINPUTS_META', NbInputs_meta ,'INPUTNAMES_META', InputNames_meta ,'INPUTSIZES_META', InputSizes_meta , 'INPUTISUSED_META', IsUsed_meta);
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

save

PURPOSE ^

SAVE a timespan object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE a timespan object to file.

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

 DESCRIPTION: SAVE overloads the save method for timespan objects.

 CALL:        Save a timespan object as a .mat file.
                >> save(ts, 'blah.mat')
              Save a timespan object as an XML file.
                >> save(ts, 'blah.xml')
              Save a timespan object to file specified in plist.
                >> save(ts, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.3 2008/02/13 16:43:14 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(timespan, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(timespan, 'Version')

 The following call returns a string that contains the routine category:

 >> category = save(timespan, 'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a timespan object into a string.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE a timespan object to file.
%
%%%
```

```
VERSION  = '$Id: save.m,v 1.3 2008/02/13 16:43:14 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'timespan') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: fsdata/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: fsdata/private   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help fsdata/private/fromxml">fsdata/private/fromxml</a>  -  read
from part of an XML tree and create a fsdata object.
   <a href="matlab:help fsdata/private/retrieve">fsdata/private/retrieve</a> -
fsdata objects with given ids from the LTPDA repository specified by the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_ss_assemble_fusion

PURPOSE ^

SystemFlatten makes a system out of assembled subsystems

SYNOPSIS ^

function [varargout] = ltpda_ss_assemble_fusion(varargin)

DESCRIPTION ^

```
 SystemFlatten makes a system out of assembled subsystems

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

 DESCRIPTION: SystemFlatten makes a system out of assembled subsystems,
 whose matrices were already put toghether is a cell array.

 CALL: [Subsys] = ltpda_ss_assemble_fusion(listMeta)

 INPUTS: listMeta - parameter list (meta-system format)

 OUTPUTS: Subsys - parameter list (subsystem format)

 PARAMETERS:
 See ltpda_ss_check function for information on subsys oupput plist content
 See ltpda_ss_assemble_arrange function for information on output plist
 content
 ***** There are no default parameter values *****%
 VERSION: $Id: ltpda_ss_assemble_fusion.m,v 1.3 2008/02/25 12:43:44 adrien Exp $

 HISTORY: 23-01-2008 A Grynagier

 TO DO : Possibly inplement vpa call for D matrix inversion. Think of
 space matrices use for inversion, and how to deal with the case when they
 are full with parameters. Check numerical precision on this step...

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ltpda_ss_check ltpda_ss_check throws an error is something is wrong is found in the subsystem.

This function is called by:

- ltpda_ss_assemble assembles embedded subsytems, with exogenous inputs
- utp_ltpda_ss_assemble_1 test function for utp_ltpda_ss_assemble using 3 subsystems, random dims.
- utp_ltpda_ss_assemble_2 test function for ltpda_ss_assemble - syntaxic

SOURCE CODE ^

```
function   [varargout] = ltpda_ss_assemble_fusion(varargin)
% SystemFlatten makes a system out of assembled subsystems
%
%%%
```

```
    return;
elseif isequal(varargin{1}, 'Category')
    varargout = CATEGORY;
    return;
end
listMeta = varargin{1};

%% retriveing fields out of plist
A_meta = find(listMeta,'A_META');
B_meta = find(listMeta,'B_META');
C_meta = find(listMeta,'C_META');
D_meta = find(listMeta,'D_META');
Bext_meta = find(listMeta,'Bext_META');
Dext_meta = find(listMeta,'Dext_META');
Ox_meta = find(listMeta,'Ox_META');
Oy_meta = find(listMeta,'Oy_META');
TIMESTEP = find(listMeta,'TIMESTEP');
Xini_meta = find(listMeta,'XINI_META');
Names_meta = find(listMeta,'NAMES_META');
ParamNames_meta = find(listMeta,'PARAMNAMES_META');
ParamValues_meta = find(listMeta,'PARAMVALUES_META');
ParamSigmas_meta = find(listMeta,'PARAMSIGMAS_META');
NbInputs_meta = find(listMeta,'NBINPUTS_META');
InputNames_meta = find(listMeta,'INPUTNAMES_META');
InputSizes_meta = find(listMeta,'INPUTSIZES_META');
InputIsUsed_meta = find(listMeta,'INPUTISUSED_META');

%% checking size of meta matrices

linesMatch = isequal(size(A_meta,1), size(B_meta,1), size(Bext_meta,1)) * isequal(size(C_meta,1), size(D_meta,1), size(Dext_meta,1)) * isequal(size(Oy_meta,1), size(Ox_meta,1) ) ;
columnsMatch = isequal(size(A_meta,2), size(C_meta,2), size(Ox_meta,2)) * isequal(size(B_meta,2), size(D_meta,2), size(Oy_meta,2)) * isequal(size(Bext_meta,2), size(Dext_meta,2), NbInputs_meta) ;
if not(linesMatch*columnsMatch)
    error('problem, wrong size in assembling meta matrices A, B, C, D, Bext, Dext, Ox, Oy');
end

%% building big meta-matrix
BigCell = [    A_meta      B_meta     Bext_meta  ;  C_meta      D_meta     Dext_meta   ;    Ox_meta     Oy_meta   cell( size(Ox_meta,1), size(Dext_meta,2) ) ] ;
% taking bounds of meta-indices
Nss_meta = size(A_meta, 1);
Nout_meta = size(Ox_meta, 1);
Imax_meta = size(BigCell,1);
Jmax_meta = size(BigCell,2);
% declaring tables for width/height of meta-columns/meta-lines
Height = zeros(Imax_meta,1);
Width = zeros(Jmax_meta,1);

%% Filling the height and width vectors for A, B, C, D, Bext and Dext
```

```
% This systematic filling is needed for sizes of A, B, C, D
for i_meta=1:Nss_meta
    Height(i_meta) = size(A_meta{i_meta,i_meta},1);
    Height(i_meta + Nss_meta) = size(C_meta{i_meta,i_meta},1);
end
for j_meta=1:Nss_meta
    Width(j_meta) = size(A_meta{j_meta,j_meta},1);
    Width(j_meta + Nss_meta) = size(C_meta{j_meta,j_meta},1);
end
% Filling the width vectors for Bext and Dext
for j_meta=1:NbInputs_meta
    Width(j_meta + 2*Nss_meta ) = InputSizes_meta(j_meta);
end
% Filling the width vectors for Ox and Oy
for i_meta=1:Nout_meta
    for j_meta=1:Nss_meta
        if not(isequal(Ox_meta{i_meta,j_meta},[]))
            Height(i_meta + 2*Nss_meta ) = size(Ox_meta{i_meta,j_meta},1);
        elseif not(isequal(Oy_meta{i_meta,j_meta},[]))
            Height(i_meta + 2*Nss_meta ) = size(Oy_meta{i_meta,j_meta},1);
        end
    end
end

%% checking their meta-size is correct in regard with their content
for i_meta=1:Imax_meta
    for j_meta=1:Jmax_meta
        if not( isequal( [], BigCell{i_meta,j_meta} ) ) % action only if cell is not empty
            %for the widht of the j meta-colums
            if not(Width(j_meta) == size(BigCell{i_meta,j_meta},2) || isempty(BigCell{i_meta,j_meta}))
                msg = 'error because matrices inside meta matrices have incompatible sizes';
                error(msg);
            end
            %same for the heigth of the i-th meta-line
            if not(Height(i_meta) == size(BigCell{i_meta,j_meta},1) || isempty(BigCell{i_meta,j_meta}))
                msg = 'error because matrices inside meta matrices have incompatible sizes';
                error(msg);
            end
        end
    end
end

%% filling the big matrix

% size of big matrix
Imax_Big = sum(Height);
Jmax_Big = sum(Width);
% taking cumulative sum of size to get to positions
```

```
CumulHeight = cumsum([1;Height]);
CumulWidth = cumsum([1;Width]);
H1 = CumulHeight(Nss_meta+1);
H2 = CumulHeight(Nss_meta*2+1);
L1 = CumulWidth(Nss_meta+1);
L2 = CumulWidth(Nss_meta*2+1);
Heigths_block = [1 H1 H2 Imax_Big+1];
Widths_block = [1 L1 L2 Jmax_Big+1];
% declaration of big matrix
Big_mat = zeros(Imax_Big, Jmax_Big);
for i_meta=1:Imax_meta
    for j_meta=1:Jmax_meta
        % taking indices for the line
        iMin = CumulHeight(i_meta);
        iMax = CumulHeight(i_meta+1)-1;
        % taking indices for the column
        jMin=CumulWidth(j_meta);
        jMax = CumulWidth(j_meta+1)-1;
        if not( isempty(BigCell{i_meta,j_meta}) || (iMin>iMax) || (jMin>jMax) )% if the local cell is not empty, and the column/line does not have zero elements
            Big_mat(iMin:iMax, jMin:jMax) = BigCell{i_meta,j_meta};
        end
    end
end

%% extraction of the submatrices out of the big matrix
nameMat = {'A' 'B' 'Bext' ; 'C' 'D' 'Dext' ; 'Ox' 'Oy' 'OO'}; % name list of the submatrices, OO is filled with zeros anyway

%% extracting the submatrices A_mat, B_mat, C_mat, D_mat, Ox_mat, Oy_mat
for I_block=1:3
    for J_block=1:2
        % meta-line/column are supposed not empty
        IisFlat=0;
        JisFlat=0;
        iMin=Heigths_block(I_block);
        iMax=Heigths_block(I_block+1)-1;
        if iMax<iMin % if the line is empty
            IisFlat=1;
        end
        jMin=Widths_block(J_block);
        jMax=Widths_block(J_block+1)-1;
        if jMax<jMin % if the column is empty
            JisFlat=1;
        end
        if IisFlat||JisFlat % if a matrix has zero elements, declare a zero matrix
            cmd = [nameMat{I_block,J_block},'_mat = zeros( iMax-iMin+1, jMax-jMin+1 ); '];
        else % other case, Extraction out of BigMat matrix.
            cmd = [ nameMat{I_block,J_block},'_mat = Big_mat(', num2str(iMin),':', num2str(iMax) ,',',  num2str(jMin),':', num2str(jMax) ,');'];
        end
```

```
        eval(cmd);
    end
end

%% extracting the submatrices Bext_mat, Dext_mat
J_block=3;
for I_block=1:2
    for j_meta = (2*Nss_meta+1):(2*Nss_meta+NbInputs_meta)
        % meta-line/column are supposed not empty
        IisFlat=0;
        JisFlat=0;
        iMin=Heigths_block(I_block);
        iMax=Heigths_block(I_block+1)-1;
        if iMax<iMin % if the line is empty
            IisFlat=1;
        end
        jMin=CumulWidth(j_meta);
        jMax=CumulWidth(j_meta+1)-1;
        if jMax<jMin % if the column is empty
            JisFlat=1;
        end
        if IisFlat||JisFlat % if a matrix has zero elements, declare a zero matrix
            cmd = [nameMat{I_block,J_block},'_mats{',num2str(j_meta-2*Nss_meta),'} = zeros( iMax-iMin+1, jMax-jMin+1 ); '];
            eval(cmd);
        else % other case, Extraction out of BigMat matrix.
            cmd = [ nameMat{I_block,J_block},'_mats{',num2str(j_meta-2*Nss_meta),'} = Big_mat(', num2str(iMin),':', num2str(iMax) ,',',  num2str(jMin),':', num2str(jMax) ,');'];
        end
        eval(cmd);
    end
end


%% take minimal system
IDinv = inv(eye(size(D_mat,1)) - D_mat); % PROBLEM HERE IF MATRIX IS SYMBOLLIC
A = A_mat + B_mat*IDinv*C_mat;
C = Ox_mat + Oy_mat*IDinv*C_mat;
BMATS = cell(size(Bext_meta, 2),1);
DMATS = cell(size(Bext_meta, 2),1);
for j_input = 1:NbInputs_meta
    BMATS{j_input} = Bext_mats{j_input} + B_mat*IDinv*Dext_mats{j_input}; %#ok<USENS>
    DMATS{j_input} = Oy_mat*IDinv*Dext_mats{j_input};
end

%% builing new initial vector and new name
Xini_mat = [];
NewName = 'SystemAssembled[';
for i=1:length(Xini_meta)
    Xini_mat = [Xini_mat ; Xini_meta{i} ]; %#ok<AGROW>
```

```
    NewName = [NewName,' ',Names_meta{i}]; %#ok<AGROW>
end
NewName = [NewName, ']'];

%% parsing output
Subsys = plist('TYPE', 'SUBSYSTEM' ,'NAME', NewName ,'TIMESTEP', TIMESTEP , ...
    'XISOUTPUT',1,'YISOUTPUT',0,'XINI', Xini_mat , ...
    'PARAMNAMES', ParamNames_meta ,'PARAMVALUES',
ParamValues_meta ,'PARAMSIGMAS', ParamSigmas_meta ,...
    'NBINPUTS', NbInputs_meta ,'INPUTNAMES', InputNames_meta ,'INPUTSIZES',
InputSizes_meta , 'INPUTISUSED', InputIsUsed_meta ,...
    'AMAT', {A} ,'BMATS', BMATS ,'CMAT', {C} ,'DMATS', DMATS );
ltpda_ss_check(Subsys);
varargout = Subsys;
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

save

PURPOSE ^

SAVE a tsdata object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE a tsdata object to file.

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

 DESCRIPTION: SAVE overloads the save method for tsdata objects.

 CALL:        Save a tsdata object as a .mat file.
                >> save(d, 'blah.mat')
              Save a tsdata object as an XML file.
                >> save(d, 'blah.xml')
              Save a tsdata object to file specified in plist.
                >> save(d, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.4 2008/02/15 16:27:16 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(tsdata, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(tsdata, 'Version')

 The following call returns a string that contains the routine category:

 >> category = save(tsdata, 'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE a tsdata object to file.
%
%%%
```

```

VERSION  = '$Id: save.m,v 1.4 2008/02/15 16:27:16 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'tsdata') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

Generated on Tue 26-Feb-2008 10:52:52 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/display">history/display</a>       -  implement
terminal display for history object.
   <a href="matlab:help history/eq">history/eq</a>            -  overloads the ==
operator for history objects.
   <a href="matlab:help history/get">history/get</a>           -  get history object
properties.
   <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/isfield">history/isfield</a>       -  tests if the
given field is one of the object properties.
   <a href="matlab:help history/ne">history/ne</a>            -  overloads the ~=
operator for history objects.
   <a href="matlab:help history/plot">history/plot</a>          -  plots a history
object as a tree diagram.
   <a href="matlab:help history/save">history/save</a>          -  a history object
to file.
   <a href="matlab:help history/set">history/set</a>           -  set a history
property.
   <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/subsref">history/subsref</a>       -  Define field
name indexing for history objects.
   <a href="matlab:help history/xml">history/xml</a>           -  convert a specwin
```

```
object to an DOM XML object.
   <a href="matlab:help history/xmladd">history/xmladd</a>        -  add an history
object to the given node of the input XML DOM
   <a href="matlab:help history/xmlparse">history/xmlparse</a>      -  creates a
history from a history xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

generic_eq

PURPOSE ^

EQ overloads the == operator for general objects.

SYNOPSIS ^

function result = generic_eq(obj1, obj2, varargin, DEFAULT_PLIST, VERSION, CATEGORY)

DESCRIPTION ^

```
 EQ overloads the == operator for general objects.

%%%
```

This function is called by:

SOURCE CODE ^

```
% EQ overloads the == operator for general objects.
%
%%%
```

```
if ~strcmp(class(obj1), class(obj2))
  disp(sprintf('\nNOT 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)
  disp(sprintf('\nNOT 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)
    field   = fields{ii};

    %% Creates the exception list for the current field.
    %% For example: {'name', 'ao/name'}
    ck_field = {field, sprintf('%s/%s', class(obj1), field)};

    % Special case (for the ao- and history-class):
    % Is the field = 'hist', 'inhists' then add 'history' to the exception list.
    %% For example: {'history', 'ao/history'}
    if ismember(field, {'hist', 'inhists'})
      ck_field{end+1} = 'history';
      ck_field{end+1} = sprintf('%s/history', class(obj1));
    elseif strcmp(field, 'val')
      ck_field{end+1} = 'value';
      ck_field{end+1} = sprintf('%s/value', class(obj1));
    end

    %% Check field if it is not in the exception list
```

```
    if ~(any(ismember(ck_field, exception_list)))

      %% Is the property an object
      if isobject(obj1(jj).(field))

        %% Check the length of the property
        if length(obj1(jj).(field)) ~= length(obj2(jj).(field))
          disp(sprintf('\nNOT EQUAL: The property [%s] of the object [%s] have not the same size', field, class(obj1)));
          result = 0;
          return
        end

        %% For each element of the property
        for kk = 1:numel(obj1(jj).(field))
          if ~eq(obj1(jj).(field)(kk), obj2(jj).(field)(kk), exception_list{:})
            result = 0;
            if numel(obj1(jj).(field)) > 1
              disp(sprintf('\nNOT EQUAL: %s.%s (%d. element)', class(obj1(jj)), field, kk));
            else
              disp(sprintf('\nNOT EQUAL: %s.%s', class(obj1(jj)), field));
            end
            return
          end
        end
      else
        if ~isequalwithequalnans(obj1(jj).(field), obj2(jj).(field))
          result = 0;
          if numel(obj1(jj)) > 1
            disp(sprintf('\nNOT EQUAL: %s.%s (%d. object)', class(obj1(jj)), field, jj));
          else
            disp(sprintf('\nNOT EQUAL: %s.%s', class(obj1(jj)), field));
          end
          return
        end
      end

    end
  end

end

```

ltpda_ss_check

PURPOSE ^

ltpda_ss_check throws an error is something is wrong is found in the subsystem.

SYNOPSIS ^

function varargout = ltpda_ss_check(varargin)

DESCRIPTION ^

```
 ltpda_ss_check throws an error is something is wrong is found in the subsystem.
%%%
```

```
 HISTORY: 21-01-2008 A Grynagier
 Creation 05-01-2008 A Grynagier

 TO DO :  Nothing for now.
 Add checks for the derviative fields later and think on how to handle the
 linear derivative case

%%%
```

```
% 'PARAMVALUES' array of doubles (means expected of parameters)
% 'PARAMSIGMAS' array of doubles (variance expected of parameters)
% 'NBINPUTS' integer number of inputs
% 'INPUTNAMES' cell array of strings (name of the inputs)
% 'INPUTSIZES'array of integers (nb of dimensions of the inputs),
% 'INPUTISUSED' double array (binary to indicate which inputs are used)
% 'AMAT' cell array (contains A matrix)
% 'BMATS' cell array (contains B matrices)
% 'CMAT' cell array (contains C matrix)
% 'DMATS' cell array (contains D matrices)
% ***** THERE ARE NO DEFAULT PARAMETERS *****
%
% VERSION: $Id: ltpda_ss_check.m,v 1.3 2008/02/25 12:43:44 adrien Exp $
%
% HISTORY: 21-01-2008 A Grynagier
% Creation 05-01-2008 A Grynagier
%
% TO DO :  Nothing for now.
% Add checks for the derviative fields later and think on how to handle the
% linear derivative case
%
%%%
```

```
    if isequal(find(syst,'NAME') , '')
        msg = 'error because subsystem has no name!';
        error(msg);
    end

    % meaningful timestep
    if find(syst,'TIMESTEP')<0
        msg = [ 'error in SubSystem ', find(syst,'NAME') , '  because negative timestep'];
        error(msg);
    end

    % check length of parameter data fields
    L4=length(find(syst,'PARAMNAMES'));
    L5=length(find(syst,'PARAMVALUES'));
    L6=length(find(syst,'PARAMSIGMAS'));
    if not( isequal(L4,L5,L6) )
        msg = [ 'error in SubSystem ', find(syst,'NAME') , ', incoherent number of parameters/means/sigmas : ', num2str(L4), ',' , num2str(L5), ',' , num2str(L6) ];
        error(msg);
    end
    % should check that the same parameter is not defined twice!!!

    % test on the number of intputs
    L7=find(syst,'NBINPUTS') ;
    L8=length( find(syst,'INPUTNAMES') );
    L9=length( find(syst,'INPUTSIZES') );
    L10=length( find(syst,'INPUTISUSED') );
    L12=length( find(syst,'BMATS') );
    L14=length( find(syst,'DMATS') );
    if not( isequal(L7,L8,L9,L10,L12,L14) )
        msg = [ 'error in SubSystem ', find(syst,'NAME') , ' incoherent number of inputs/names/sizes/used/Bmats/Dmats: ', num2str(L7), ',' , num2str(L8), ',' , num2str(L9), ',' , num2str(L10), ',' , num2str(L12), ',' , num2str(L14) ];
        error(msg);
    end

    % test on the size of intput matrices
    % and fitting to state space and output size
    AMAT = find(syst,'AMAT');
    CMAT = find(syst,'CMAT');
    SSsize = size(AMAT{1},1);
    OUTsize = size(CMAT{1},1);
    INPUTSIZES = find(syst,'IMPUTNAMES');
    BMATS = find(syst,'BMATS');
    DMATS = find(syst,'DMATS');
    INPUTNAMES = find(syst,'INPUTNAMES');
    for i=1:length(INPUTSIZES)
        if not( isequal(INPUTSIZES(i),size(BMATS{i},2),size(DMATS{i},2) ) )
            msg = [ 'error in SubSystem ', find(syst,'NAME') , ' for input named ', INPUTNAMES{i} ,' because size of input matrices and declared size do not match altoghether'];
            error(msg);
```

```
        end
        if not( isequal(SSsize,size(BMATS{i},1)) )
            msg = [ 'error in SubSystem ', find(syst,'NAME') , ' because wrong size of B matrix (height does not fit to State Space) in input named ', INPUTNAMES{i}];
            error(msg);
        end
        if not( isequal(OUTsize,size(DMATS{i},1)) )
            msg = [ 'error in SubSystem ', find(syst,'NAME') , ' because wrong size of D matrix (height does not fit to Output) in input named ', INPUTNAMES{i}];
            error(msg);
        end
    end

    % test on A and C
    if not( isequal(SSsize, size(AMAT{1},2), size(CMAT{1},2)) )
        msg= [ 'error in SubSystem ', find(syst,'NAME') , ' either because A is not square or C does not fit to A'];
        error(msg);
    end
catch
    varargout =0;
end

end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

save

PURPOSE ^

SAVE an xydata object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE an xydata object to file.

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

 DESCRIPTION: SAVE overloads the save method for xydata objects.

 CALL:        Save an xydata object as a .mat file.
                >> save(d, 'blah.mat')
              Save an xydata object as an XML file.
                >> save(d, 'blah.xml')
              Save an xydata object to file specified in plist.
                >> save(d, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.4 2008/02/15 16:27:16 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(xydata, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(xydata, 'Version')

 The following call returns a string that contains the routine category:

 >> category = save(xydata, 'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE an xydata object to file.
%
%%%
```

```

VERSION  = '$Id: save.m,v 1.4 2008/02/15 16:27:16 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'xydata') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: history/private %%%%%%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%   class: history/private   %%%%%%%%%%%%%%%%%%%%

   <a href="matlab:help history/private/fromxml">history/private/fromxml</a>  -
read from part of an XML tree and create a history object.
   <a href="matlab:help history/private/retrieve">history/private/retrieve</a> -
history objects with given ids from the LTPDA repository specified by the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

generic_get

PURPOSE ^

GET get a property of a general object.

SYNOPSIS ^

function val = generic_get(obj, prop_name, DEFAULT_PLIST, VERSION, CATEGORY)

DESCRIPTION ^

```
 GET get a property of a general object.

%%%
```

```
% HISTORY:     11-02-2008 Diepholz
%                Creation
%
%%%
```

ltpda_ss_iir2ss

PURPOSE ^

ltpda_miir2ss converts a iir filter model into a subsystem.

SYNOPSIS ^

function varargout = ltpda_ss_iir2ss(varargin)

DESCRIPTION ^

```
 ltpda_miir2ss converts a iir filter model into a subsystem.
%%%
```

```
 'INPUTISUSED' double array (binary to indicate which inputs are used)
 'AMAT' cell array (contains A matrix)
 'BMATS' cell array (contains B matrices)
 'CMAT' cell array (contains C matrix)
 'DMATS' cell array (contains D matrices)
 ***** THERE ARE NO DEFAULT PARAMETERS *****

 VERSION: $Id: ltpda_ss_iir2ss.m,v 1.1 2008/02/25 12:43:44 adrien Exp $

 HISTORY: 8-02-2008 A Grynagier
 7-02-2008 A Grynagier
 2-02-2008 A Grynagier

 TO DO :  possibly add the jordan block option

%%%
```

```
% ltpda_ss_check function):
% plist('TYPE', TYPE ,'NAME', NAME ,'TIMESTEP', TIMESTEP ,...
%     'PARAMNAMES', PARAMNAMES ,'PARAMVALUES', PARAMVALUES ,'PARAMSIGMAS', PARAMSIGMAS ,...
%     'NBINPUTS', NBINPUTS ,'INPUTNAMES', INPUTNAMES ,'INPUTSIZES', INPUTSIZES ,'INPUTISUSED', INPUTISUSED ,...
%     'AMAT', AMAT ,'BMATS', BMATS ,'CMAT', CMAT ,'DMATS', DMATS );% PARAMETERS:
% 'TYPE' should be 'SUBSYSTEM'
% 'NAME' is a string
% 'TIMESTEP' is a real positive integer, set to zero in continuous case
% 'PARAMNAMES' cell array of strings describing parameter variables
% 'PARAMVALUES' array of doubles (means expected of parameters)
% 'PARAMSIGMAS' array of doubles (variance expected of parameters)
% 'NBINPUTS' integer number of inputs
% 'INPUTNAMES' cell array of strings (name of the inputs)
% 'INPUTSIZES'array of integers (nb of dimensions of the inputs),
% 'INPUTISUSED' double array (binary to indicate which inputs are used)
% 'AMAT' cell array (contains A matrix)
% 'BMATS' cell array (contains B matrices)
% 'CMAT' cell array (contains C matrix)
% 'DMATS' cell array (contains D matrices)
% ***** THERE ARE NO DEFAULT PARAMETERS *****
%
% VERSION: $Id: ltpda_ss_iir2ss.m,v 1.1 2008/02/25 12:43:44 adrien Exp $
%
% HISTORY: 8-02-2008 A Grynagier
% 7-02-2008 A Grynagier
% 2-02-2008 A Grynagier
%
% TO DO :  possibly add the jordan block option
%
%%%
```

```
b = iir.b;

% for next part check must be made there is no pole zero cancelation
nss = size(b,2)-1;

% is residue ok??
[r,p,k] = residue(a,b); %!! here b are for the poles, a the zeros
D = k;

a = a/b(1);
b = b/b(1);


A =  [zeros(nss-1,1) eye(nss-1); b(2:length(b))];
B = [zeros(nss-1,1) ;1];
C = a;


varargout = plist('TYPE', 'SUMBSYSTEM' ,'NAME', iir.name ,'TIMESTEP', 1/(iir.fs), ...
    'XISOUTPUT', 0,'YISOUTPUT',1,'XINI', zeros(nss,1) , ...
    'PARAMNAMES', {},'PARAMVALUES', [],'PARAMSIGMAS', [],...
    'NBINPUTS', iir.ntaps ,'INPUTNAMES', {'U_1'} ,'INPUTSIZES', 1 ,'INPUTISUSED', 1 ,...
    'AMAT', {A} ,'BMATS', {B} ,'CMAT', {C} ,'DMATS', {D} );

ltpda_ss_check(subsys);
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

save

PURPOSE ^

SAVE an xyzdata object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE an xyzdata object to file.

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

 DESCRIPTION: SAVE overloads the save method for xyzdata objects.

 CALL:        Save an xyzdata object as a .mat file.
                >> save(d, 'blah.mat')
              Save an xyzdata object as an XML file.
                >> save(d, 'blah.xml')
              Save an xyzdata object to file specified in plist.
                >> save(d, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.4 2008/02/13 13:46:45 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(xyzdata, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(xyzdata, 'Version')

 The following call returns a string that contains the routine category:

 >> category = save(xyzdata, 'Category')

 HISTORY: 24-12-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE an xyzdata object to file.
%
%%%
```

```
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: mfir %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: mfir   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help mfir/char">mfir/char</a>     -  convert an mfir object into
a string.
   <a href="matlab:help mfir/display">mfir/display</a>  -  overloads display
functionality for mfir objects.
   <a href="matlab:help mfir/eq">mfir/eq</a>       -  overloads the == operator for
mfir objects.
   <a href="matlab:help mfir/get">mfir/get</a>      -  get mfir properties.
   <a href="matlab:help mfir/isfield">mfir/isfield</a>  -  tests if the given field
is one of the object properties.
   <a href="matlab:help mfir/mfir">mfir/mfir</a>     -  FIR filter object class
constructor.
   <a href="matlab:help mfir/ne">mfir/ne</a>       -  overloads the ~= operator for
mfir objects.
   <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/save">mfir/save</a>     -  an mfir object to file.
   <a href="matlab:help mfir/set">mfir/set</a>      -  set a mfir object property.
   <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/subsref">mfir/subsref</a>  -  Define field name
indexing for mfir objects.
   <a href="matlab:help mfir/xml">mfir/xml</a>      -  convert an mfir object to an
DOM XML object.
   <a href="matlab:help mfir/xmladd">mfir/xmladd</a>   -  add an MFIR object to the
given node of the input XML DOM
   <a href="matlab:help mfir/xmlparse">mfir/xmlparse</a> -  creates a mfir filter
from a mfir xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

generic_set

PURPOSE ^

SET set a property of a general object.

SYNOPSIS ^

function objs = generic_set(objs, propArgin, DEFAULT_PLIST, VERSION, CATEGORY)

DESCRIPTION ^

```
 SET set a property of a general object.

%%%
```

```
% HISTORY:     11-02-2008 Diepholz
%                 Creation
%
%%%
```

```
  end
end

while length(propArgin) >= 2

  prop      = propArgin{1};
  val       = propArgin{2};
  propArgin = propArgin(3:end);

  %%% For all objects
  for jj = 1:numel(objs)
    try
      objs(jj).(prop) = val;
    catch
      error('### ''%s'' is not a valid %s-object property.', prop, class(objs));
    end

  end
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_ss_kalman_filter

PURPOSE ^

Kalman filter

SYNOPSIS ^

function varargout = ltpda_ss_kalman_filter(varargin)

DESCRIPTION ^

```
Kalman filter
%%%
```

```
%                                'MEAN' (vector) and a sigma 'SIGMA' (Matrix)
% syst - system data in plist format
%
% OUTPUTS: ao_out _ output analysis object
%
% ***** THERE ARE NO DEFAULT PARAMETERS *****
%
% &&VERSION: $Id: ltpda_ss_kalman_filter.m,v 1.2 2008/02/25 12:43:44 adrien Exp $
%
% HISTORY: 19-02-2008 A Grynagier
% 17-02-2008 A Grynagier
% Creation 30-01-2008 M Weyrich
%
% TO DO: options to be defined
% implement input type of three kinds:
% - AO
% - Constant
% - gaussian white noise
% use LTPDA functions to generate white noise
%%%
```

```
Xini = find(syst, 'XINI');
timestep = find(syst, 'TIMESTEP');
NBINPUTS = find(syst, 'NBINPUTS');
INPUTISUSED = find(syst, 'INPUTISUSED');
INPUTNAMES = find(syst, 'INPUTNAMES');
% INPUTSIZES = find(syst, 'INPUTSIZES');
XISOUTPUT = find(syst, 'XISOUTPUT');


%% get info from plist for noise inputs
% input covariances
Qs = find(data_plist, 'Qs');
Rs = find(data_plist, 'Rs');
Noise_Names = find(data_plist, 'Noise_Names');
Q = zeros(nss);
R = zeros(nout);
for i=1:length(Noise_Names)
    for j=1:NBINPUTS
        if isequal(Noise_Names{i},INPUTNAMES{j} )
            Q = Q + B{j}*Qs{i}*B{j}'; %assumption in this covariance addition is that the inputs are not correlated
            R = R + D{j}*Rs{i}*D{j}';
        end
    end
end

%% read out AO (time-series data)
ao_in = cell(NBINPUTS,1);
for i=1:NBINPUTS
    if INPUTISUSED(i)
        ao_in{i} = find(ao_in_list, INPUTNAMES{i});
        if isequal(class(ao_in{i}), 'ao')
            if not(isequal(ao_in{i}.data.x,{})) %making sure ao is not empty
                t = ao_in{i}.data.x;
                dt = diff(t);
                if sum(not(isequal(dt,timestep)))>1
                    msg = ['error in input number ', i , ' called ', INPUTNAMES{i}, 'because wrong timestep for simulation'];
                    error(msg);
                end
                if not( isequal( size(B{i},1), size(D{i},1), size(ao_in{i}.data.y,1) ) )
                    msg = ['error in input number ', i , ' called ', INPUTNAMES{i}, 'because wrong input size'];
                    error(msg);
                end
            end
        end
    end
end
%% getting correct number of samples
nsamples = size(ao_Y.data.y,1);

```

```
%% taking only output data (in case XISOUTPUT = 1)
SizeData = size(ao_Y.data.y,2);
if XISOUTPUT
    range = (nss+1):SizeData;
else
    range = 1:SizeData;
end




%% Kalman filter vector allocation
P0=eye(nss);
X_est=zeros(size(A,1),nsamples+1);
X_upd=zeros(size(A,1),nsamples+1);
X_upd(:,1)=Xini;
Y_tilde=zeros(nout,nsamples+1);

%%%%%%%%%%%%%%%%%%
%%% Kalman filter %% chg
%%%%%%%%%%%%%%%%%%

for i=2:nsamples %chg
    IncrementX = zeros(nss,1);
    IncrementY = zeros(nout,1);
    for j=1:length(ao_in)
        if not(isequal(ao_in{j},[]))
            IncrementX = IncrementX +  B{j}*ao_in{j}.data.y(i,:) ;
            IncrementY = IncrementY +  D{j}*ao_in{j}.data.y(i,:) ;
        end
    end
    X_est(:,i) = A*X_upd(:,i-1)+IncrementX; %Estimate
    Y_tilde(:,i) = ao_Y.data.y(i,range)'-C*X_est(:,i)-IncrementY; %Measurement residual chg dEst forgotten
    P = A*P0*A'+Q;
    S = C*P*C'+R;  %Covariance of Y-tilde
    K = P*C'*inv(S); %K. gain
    X_upd(:,i) = X_est(:,i)+K*Y_tilde(:,i); %Update
    P0 = (eye(size(K,1))-K*C)*P;
end


%% data object
if exist('t','var')
    ts = tsdata(t,[X_est; Y_tilde]) ;
else
    ts = tsdata(timestep*(1:(nsamples+1))',[X_est; Y_tilde]') ;
end

%% history object, old history from ao_in plus new
% hi_old = ao_Y.hist;
hi_new = history('Kalman','version', syst);
% hi = [hi_old hi_new];
```

```
% h = history(hi);

%% construct output analysis object
varargout  = ao(ts,hi_new);
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

save

PURPOSE ^

SAVE a zero object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE a zero object to file.

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

 DESCRIPTION: SAVE overloads the save method for zero objects.

 CALL:        Save a zero object as a .mat file.
                >> save(z, 'blah.mat')
              Save a zero object as an XML file.
                >> save(z, 'blah.xml')
              Save a zero object to file specified in plist.
                >> save(z, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.4 2008/02/15 16:27:16 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(zero, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(zero, 'Version')

 The following call returns a string that contains the routine category:

 >> category = save(zero, 'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a zero object into a string.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE a zero object to file.
%
%%%
```

```

ALGONAME = mfilename;
VERSION  = '$Id: save.m,v 1.4 2008/02/15 16:27:16 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'zero') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: mfir/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%   class: mfir/private   %%%%%%%%%%%%%%%%%%

   <a href="matlab:help mfir/private/fromxml">mfir/private/fromxml</a>           -
read from part of an XML tree and create an mfir object.
   <a href="matlab:help mfir/private/mkbandpass">mfir/private/mkbandpass</a>
-  return a bandpass filter miir(). A Cheby filter is used.
   <a href="matlab:help mfir/private/mkbandreject">mfir/private/mkbandreject</
a>      -  return a low pass filter miir(). A Butterworth filter is used.
   <a href="matlab:help mfir/private/mkhighpass">mfir/private/mkhighpass</a>
-  return a high pass filter miir(). A Butterworth filter is used.
   <a href="matlab:help mfir/private/mklowpass">mfir/private/mklowpass</a>
-  return a low pass filter mfir().
   <a href="matlab:help mfir/private/parseFilterParams">mfir/private/
parseFilterParams</a> -  parses the input plist and returns a full plist for
   <a href="matlab:help mfir/private/retrieve">mfir/private/retrieve</a>          -
mfir objects with given ids from the LTPDA repository specified by the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_ss_make

PURPOSE ^

ltpda_ss_make is the generic function to decalre or load a subsystem.

SYNOPSIS ^

function varargout = ltpda_ss_make(varargin)

DESCRIPTION ^

```
 ltpda_ss_make is the generic function to decalre or load a subsystem.
%%%
```

```
 'INPUTISUSED' double array (binary to indicate which inputs are used)
 'AMAT' cell array (contains A matrix)
 'BMATS' cell array (contains B matrices)
 'CMAT' cell array (contains C matrix)
 'DMATS' cell array (contains D matrices)
 ***** THERE ARE NO DEFAULT PARAMETERS *****

 VERSION: $Id: ltpda_ss_make.m,v 1.3 2008/02/25 12:43:44 adrien Exp $

 HISTORY: 18-02-2008 A Grynagier
 22-01-2008 A Grynagier
 Creation 02-01-2008 A Grynagier

 TO DO : finnish and check the iir and plist parts - see also test function
%%%
```

```
% ltpda_ss_check function):
% plist('TYPE', TYPE ,'NAME', NAME ,'TIMESTEP', TIMESTEP ,...
%     'PARAMNAMES', PARAMNAMES ,'PARAMVALUES', PARAMVALUES ,'PARAMSIGMAS', PARAMSIGMAS ,...
%     'NBINPUTS', NBINPUTS ,'INPUTNAMES', INPUTNAMES ,'INPUTSIZES', INPUTSIZES ,'INPUTISUSED', INPUTISUSED ,...
%     'AMAT', AMAT ,'BMATS', BMATS ,'CMAT', CMAT ,'DMATS', DMATS );% PARAMETERS:
% 'TYPE' should be 'SUBSYSTEM'
% 'NAME' is a string
% 'TIMESTEP' is a real positive integer, set to zero in continuous case
% 'PARAMNAMES' cell array of strings describing parameter variables
% 'PARAMVALUES' array of doubles (means expected of parameters)
% 'PARAMSIGMAS' array of doubles (variance expected of parameters)
% 'NBINPUTS' integer number of inputs
% 'INPUTNAMES' cell array of strings (name of the inputs)
% 'INPUTSIZES'array of integers (nb of dimensions of the inputs),
% 'INPUTISUSED' double array (binary to indicate which inputs are used)
% 'AMAT' cell array (contains A matrix)
% 'BMATS' cell array (contains B matrices)
% 'CMAT' cell array (contains C matrix)
% 'DMATS' cell array (contains D matrices)
% ***** THERE ARE NO DEFAULT PARAMETERS *****
%
% VERSION: $Id: ltpda_ss_make.m,v 1.3 2008/02/25 12:43:44 adrien Exp $
%
% HISTORY: 18-02-2008 A Grynagier
% 22-01-2008 A Grynagier
% Creation 02-01-2008 A Grynagier
%
% TO DO : finnish and check the iir and plist parts - see also test function
%%%
```

```
% for ii = 1:nargin
%   in_names{end+1} = inputname(ii);
% end
%
% [as, ps, invars] = collect_inputs(varargin, in_names);
%
% ps = combine(ps, getDefaultPlist);
%
% subsys = plist(ps.params.key, eval(ps.params.val));



%% proceeding data
if ischar(varargin{1})
    pathbelow = 'LTPDA_SS_GEN_1\';
    cmd1 = ['struct = load( ''',pathbelow, varargin{1} , ''' ); '];

    cmd2 = ['subsys = struct.', varargin{1} , '; '];
    eval(cmd1);
    eval(cmd2);
    ltpda_ss_check(subsys); %#ok<NODEF>
elseif isequal(class(varargin{1}), 'plist object')
    subsys = varargin{1};
    ltpda_ss_check(subsys);
elseif isequal(class(varargin{1}), 'pzmodel')
    pzm = varargin{1};
    display('not fully implemented yet');
    subsys = ltpda_ss_pz2ss(pzm);
elseif isequal(class(varargin{1}), 'miir')
    iir = varargin{1};
    subsys = ltpda_ss_iir2ss(iir);
else
    error('wrong type of input: not a string nor a plist object');
end
ltpda_ss_check(subsys);
varargout = subsys;
end



```

Generated on Tue 26-Feb-2008 10:52:52 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/display">miir/display</a>  -  overloads display
functionality for miir objects.
   <a href="matlab:help miir/eq">miir/eq</a>       -  overloads the == operator for
miir objects.
   <a href="matlab:help miir/get">miir/get</a>      -  get miir properties.
   <a href="matlab:help miir/isfield">miir/isfield</a>  -  tests if the given field
is one of the object properties.
   <a href="matlab:help miir/miir">miir/miir</a>     -  IIR filter object class
constructor.
   <a href="matlab:help miir/ne">miir/ne</a>       -  overloads the ~= operator for
miir objects.
   <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/save">miir/save</a>     -  an miir object to file.
   <a href="matlab:help miir/set">miir/set</a>      -  set a miir object property.
   <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/subsref">miir/subsref</a>  -  Define field name
indexing for miir objects.
   <a href="matlab:help miir/xml">miir/xml</a>      -  convert an miir object to an
DOM XML object.
   <a href="matlab:help miir/xmladd">miir/xmladd</a>   -  add an MIIR object to the
given node of the input XML DOM
   <a href="matlab:help miir/xmlparse">miir/xmlparse</a> -  creates a miir filter
from a miir xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get csdata properties.

SYNOPSIS ^

function val = get(c, propName)

DESCRIPTION ^

```
 GET get csdata properties.

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

 DESCRIPTION: GET get csdata properties.

 CALL:        name    = get(c, 'name');
              created = get(c, 'created');
              version = get(c, 'version');
              y       = get(c, 'y');
              y       = get(c, 'vals');
              x       = get(c, 'x');
              x       = get(c, 'tags');
              xunit   = get(c, 'xunits');
              yunit   = get(c, 'yunits');

 VERSION:     $Id: get.m,v 1.7 2008/02/11 17:28:45 ingo Exp $

 HISTORY:     30-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- char CHAR convert a cdata object into a string.
- display DISPLAY implement terminal display for cdata object.

SOURCE CODE ^

```
function val = get(c, propName)
% GET get csdata properties.
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_ss_modify

PURPOSE ^

paramSubstitute enables to modify parameters and retrieve numerical

SYNOPSIS ^

function varargout = ltpda_ss_modify(varargin)

DESCRIPTION ^

```
 paramSubstitute enables to modify parameters and retrieve numerical
 values

%%%
```

```
% paramSubstitute enables to modify parameters and retrieve numerical
% values
%
%%%
```

```
% differenciation
%
% PARAMETERS:
% ================ 'modif' field ================
% A: ...
% ================ 'derivatives' field ================
% ================ 'derivatives2' field ================
%
% VERSION: $Id: ltpda_ss_modify.m,v 1.3 2008/02/25 12:43:44 adrien Exp $
%
% HISTORY: 13-02-2008 A Grynagier
% 29-01-2008 A Grynagier
% Creation
%
% % TO DO :
% update help header (orders 'ALL', 'NO')
% display in window parametric substitutions and differenciation proceeded
% Define user scenarii to see ensure output is satisfying
% Various numerical checks

%% standard calls for LTPDA function data

ALGONAME = mfilename;
VERSION = '$Id: ltpda_ss_modify.m,v 1.3 2008/02/25 12:43:44 adrien Exp $';
CATEGORY = 'STATESPACE';
display(['starting ' ALGONAME]);

if isequal( varargin{1}, 'Version')
    varargout = VERSION;
    return;
elseif isequal(varargin{1}, 'Params')
    varargout = plist( 'NOT_USED_PARAM_NAMES', {'param1', 'param2'}, 'NOT_USED_PARAM_ORDER', [1 0 ],...
        'SET_PARAM_NAMES', {'param2', 'param3'}, 'SET_PARAM_VALUES', [3 6], 'SET_PARAM_SIGMAS', [0.2 0.1],...
        'SET_PARAM_VALUES_ORDER', 'ALL', 'SET_PARAM_SIGMAS_ORDER', 'NO', ...
        'JAC_PARAM_NAMES', {'param2', 'param4'}, 'JAC_PARAM_ORDER', [1 1],...
        'HESS_PARAM_NAMES', {'param2', 'param3'}, 'HESS_PARAM_ORDER' , [1 0],...
    'SYMBOLIC_OUTPUT_ALLOWED', 1);
    return;
elseif isequal(varargin{1}, 'Category')
    varargout = CATEGORY;
    return;
end

%% Retriveing option fields for operations

if length(varargin)==2
    System = varargin{1};
    modif = varargin{2};
    %add loop in case modif is an array of plists
else
    error('wrong number of inputs!')
```

```
end
% fields from user options
NotUsedParamNamesIni = find(modif,'NOT_USED_PARAM_NAMES') ;
NotUsedParamOrder = find(modif,'NOT_USED_PARAM_ORDER') ;
SetParamNamesIni = find(modif,'SET_PARAM_NAMES') ;
SetParamValuesIni = find(modif,'SET_PARAM_VALUES') ;
SetParamSigmasIni = find(modif,'SET_PARAM_SIGMAS') ;
SetParamValuesOrder = find(modif,'SET_PARAM_VALUES_ORDER') ;
SetParamSigmasOrder = find(modif,'SET_PARAM_SIGMAS_ORDER') ;
JacParamNamesIni = find(modif,'JAC_PARAM_NAMES') ;
JacParamOrder = find(modif,'JAC_PARAM_ORDER') ;
HessParamNamesIni = find(modif,'HESS_PARAM_NAMES') ;
HessParamOrder = find(modif,'HESS_PARAM_ORDER') ;
SymbolicOutputAllowed = find(modif,'SYMBOLIC_OUTPUT_ALLOWED') ;

% subsystem data matrices
SystemAMat = find(System,'AMAT');
SystemBMats = find(System,'BMATS');
SystemCMat = find(System,'CMAT');
SystemDMats = find(System,'DMATS');
% subsystem parametric data
SystemParamNames = find(System,'PARAMNAMES');
SystemParamValues = find(System,'PARAMVALUES');
SystemParamSigmas = find(System,'PARAMSIGMAS');
SystemNbParams = length(SystemParamNames);

%% building updated list for NotUsedParam...
if isequal(NotUsedParamOrder,'ALL')
    NotUsedParamNames = NotUsedParamNamesIni;
elseif isequal(NotUsedParamOrder,'NO')
    NotUsedParamNames = {};
else
    NotUsedParamNames = {};
    j=1;
    for i=1:length(NotUsedParamNamesIni)
        if NotUsedParamOrder(i)
            NotUsedParamNames{j} = NotUsedParamNamesIni{i}; %#ok<AGROW>
            j = j+1;
        end
    end
end
NotUsedParamValues = [];
for i=1:length(NotUsedParamNames)
    for j=1:length(SystemParamNames)
        if isequal(NotUsedParamNames{i},SystemParamNames{j})
            NotUsedParamValues(i)=SystemParamValues(j); %#ok<AGROW>
        end
    end
end
%% building updated list for SetParam...
if (isequal(SetParamValuesOrder,'ALL'))
    SetParamValuesOrder = ones(length(SetParamNamesIni),1);
elseif (isequal(SetParamValuesOrder,'NO'))
```

```
    SetParamValuesOrder = zeros(length(SetParamNamesIni),1);
end
if (isequal(SetParamSigmasOrder,'ALL'))
    SetParamSigmasOrder = ones(length(SetParamNamesIni),1);
elseif (isequal(SetParamSigmasOrder,'NO'))
    SetParamSigmasOrder = zeros(length(SetParamNamesIni),1);
end
SetParamNames = {};
SetParamValues = [];
SetParamSigmas = [];
j =1;
for i=1:length(SetParamNamesIni)
    if SetParamValuesOrder(i) || SetParamSigmasOrder(i)
        SetParamNames{j} = SetParamNamesIni{i}; %#ok<AGROW>
        if SetParamValuesOrder(i)
            SetParamValues(j) = SetParamValuesIni(i); %#ok<AGROW>
        else
            for k=1:SystemNbParams
                if isequal(SystemParamNames{k}, SetParamNamesIni{i})
                    SetParamValues(j) = SystemParamValues(k); %#ok<AGROW>
                end
            end
        end
        if   SetParamSigmasOrder(i)
            SetParamSigmas(j) = SetParamSigmasIni(i); %#ok<AGROW>
        else
            for k=1:SystemNbParams
                if isequal(SystemParamNames{k}, SetParamNamesIni{i})
                    SetParamSigmas(j) = SystemParamSigmas(k); %#ok<AGROW>
                end
            end
        end
        j = j+1;
    end
end
%% building updated list for JacParam...
if isequal(JacParamOrder,'ALL')
    JacParamNames = JacParamNamesIni;
elseif isequal(JacParamOrder,'NO')
    JacParamNames = {};
else
    JacParamNames = {};
    j=1;
    for i=1:length(JacParamNamesIni)
        if JacParamOrder(i)
            JacParamNames{j} = JacParamNamesIni{i}; %#ok<AGROW>
            j = j+1;
        end
    end
end
%% building updated list for HessParam...
if isequal(HessParamOrder,'ALL')
    HessParamNames = HessParamNamesIni;
```

```
elseif isequal(HessParamOrder,'NO')
    HessParamNames = {};
else
    HessParamNames = {};
    j=1;
    for i=1:length(HessParamNamesIni)
        if HessParamOrder(i)
            HessParamNames{j} = HessParamNamesIni{i}; %#ok<AGROW>
            j = j+1;
        end
    end
end
nNotUsed = length(NotUsedParamNames);
nSet = length(SetParamNames);
nJac = length(JacParamNames);
nHess = length(HessParamNames);

%% building param position tables
InSet = zeros(SystemNbParams,1);
InJac = zeros(SystemNbParams,1);
InHess = zeros(SystemNbParams,1);
InNotUsed = zeros(SystemNbParams,1);
for i=1:SystemNbParams
    for j=1:nNotUsed
        if isequal(SystemParamNames{i},NotUsedParamNames{j})
            InNotUsed(i) = j;
        end
    end
    for j=1:nSet
        if isequal(SystemParamNames{i},SetParamNames{j})
            InSet(i) = j;
        end
    end
    for j=1:nJac
        if isequal(SystemParamNames{i},JacParamNames{j})
            InJac(i) = j;
        end
    end
    for j=1:nHess
        if isequal(SystemParamNames{i},HessParamNames{j})
            InHess(i) = j;
        end
    end
end

%% looking for non exisitng but modified parameter names
if not(length(NotUsedParamNames)==sum(InNotUsed>0))
    msg = 'error with notusedparameter list because one does not exist in system.';
    error(msg);
elseif not(length(SetParamNames)==sum(InSet>0))
    msg = 'error with setparameter list because one does not exist in system.';
    error(msg);
```

```
elseif not(length(JacParamNames)==sum(InJac>0))
    msg = 'error with jacparameter list because one does not exist in system.';
    error(msg);
elseif not(length(HessParamNames)==sum(InHess>0))
    msg = 'error with hessparameter list because one does not exist in system.';
    error(msg);
end

%% looking of unused but used parameters
for i=1:nNotUsed
    for j =1:nSet
        if isequal(NotUsedParamNames{i},SetParamNames{j})
            msg = 'error because parameter set was supposed to be unused';
            error(msg);
        end
    end
    for j =1:nJac
        if isequal(NotUsedParamNames{i},JacParamNames{j})
            msg = 'error because parameter for jacobian was supposed to be unused';
            error(msg);
        end
    end
    for j =1:nHess
        if isequal(NotUsedParamNames{i},HessParamNames{j})
            msg = 'error because parameter for hessian was supposed to be unused';
            error(msg);
        end
    end
end

%% Substituting unused parameters
for i=1:nNotUsed
    if InNotUsed(i)>0
        SystemAMat{1} = subs(SystemAMat{1}, NotUsedParamNames(InNotUsed(i)) , NotUsedParamValues(InNotUsed(i)));
        SystemCMat{1} = subs(SystemCMat{1}, NotUsedParamNames(InNotUsed(i)) , NotUsedParamValues(InNotUsed(i)));
        for j=1:length(SystemBMats)
            SystemBMats{j} = subs(SystemBMats{j}, NotUsedParamNames(InNotUsed(i)) , NotUsedParamValues(InNotUsed(i)));
            SystemDMats{j} = subs(SystemDMats{j}, NotUsedParamNames(InNotUsed(i)) , NotUsedParamValues(InNotUsed(i)));
        end
    end
end

%% modifying parameter tables in System (setparam list)
for i=1:SystemNbParams
    if InSet(i)>0
        SystemParamValues(i) = SetParamValues(InSet(i));
        SystemParamSigmas(i) = SetParamSigmas(InSet(i));
```

```
    end
end
%% Choosing parameter lists for substitution
if not(SymbolicOutputAllowed)
    display('Forbidden symbollic output case is not handled in the current version' );
    SubsNames = SystemParamNames;
    SubsValues = SystemParamValues;
else
    SubsNames = SetParamNames;
    SubsValues = SetParamValues;
end


%% building jacobians - symbollic expression
% creating symbollic variables vector
if nJac>0
    JacParamVar = zeros(nJac,1);
    for i = 1:length(JacParamNames2)
        nameVar = eval(['JacParamNames{', num2str(i),'}']);
        cmd = [nameVar ,'=sym(', nameVar ,');']; %Declaration not needed !!
        eval(cmd);
        cmd2 = ['JacParamVar(',num2str(i),')=',nameVar];
        eval(cmd2);
    end
    % differenciating
    SystemAMat{1,2} = subs( ltpda_jacobian(SystemAMat{1}, JacParamVar,1), SubsNames , SubsValues);
    SystemCMat{1,2} = subs( ltpda_jacobian(SystemCMat{1}, JacParamVar,1), SubsNames , SubsValues);
    for j=1:length(SystemBMats)
        SystemBMats{j,2} = subs( ltpda_jacobian(SystemBMats{j}, JacParamVar,1), SubsNames , SubsValues);
        SystemDMats{j,2} = subs( ltpda_jacobian(SystemDMats{j}, JacParamVar,1), SubsNames , SubsValues);
    end
end

%% building hessians - symbollic expression

% creating symbollic variables vector
if nHess>0
    HessParamVar = zeros(nHess,1);
    for j = 1:nHess
        nameVar = eval(['HessParamNames2{', num2str(j),'}']);
        cmd = [nameVar ,'=sym(', nameVar ,');']; %Declaration not needed !!
        eval(cmd);
        cmd2 = ['HessParamVar(',num2str(j),')=',nameVar];
        eval(cmd2);
    end
    % differenciating
    SystemAMat{1,3} = subs( ltpda_jacobian(SystemAMat{1}, HessParamVar,2),
```

```
SubsNames , SubsValues);
    SystemCMat{1,3} = subs( ltpda_jacobian(SystemCMat{1}, HessParamVar,2), SubsNames , SubsValues);
    for j=1:length(SystemBMats)
        SystemBMats{j,3} = subs( ltpda_jacobian(SystemBMats{j}, HessParamVar,2), SubsNames , SubsValues);
        SystemDMats{j,3} = subs( ltpda_jacobian(SystemDMats{j}, HessParamVar,2), SubsNames , SubsValues);
    end
end

%% Numerical value for system

if nSet>0
    SystemAMat{1,1} = subs(SystemAMat{1}, SubsNames , SubsValues);
    SystemCMat{1,1} = subs(SystemCMat{1}, SubsNames , SubsValues);
    for j=1:length(SystemBMats)
        SystemBMats{j,1} = subs(SystemBMats{j}, SubsNames , SubsValues);
        SystemDMats{j,1} = subs(SystemDMats{j}, SubsNames , SubsValues);
    end
end

%% parsing into new system
varargout = pset(system, 'AMAT',SystemAMat, 'BMATS', SystemBMats, 'CMAT', SystemCMat, 'DMATS', SystemDMats);
% to be added  : modyfiying the parameter lists!

end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: miir/private %%%%%%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: miir/private   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help miir/private/filload">miir/private/filload</a>           -
a LISO *_iir.fil file to get the filter taps and return a
   <a href="matlab:help miir/private/fromxml">miir/private/fromxml</a>           -
read from part of an XML tree and create an miir object.
   <a href="matlab:help miir/private/mfiltload">miir/private/mfiltload</a>
-  Load in a filter from *.filt files.
   <a href="matlab:help miir/private/mkbandpass">miir/private/mkbandpass</a>
-  return a bandpass filter miir(). A Cheby filter is used.
   <a href="matlab:help miir/private/mkbandreject">miir/private/mkbandreject</
a>      -  return a low pass filter miir(). A Butterworth filter is used.
   <a href="matlab:help miir/private/mkhighpass">miir/private/mkhighpass</a>
-  return a high pass filter miir(). A Butterworth filter is used.
   <a href="matlab:help miir/private/mklowpass">miir/private/mklowpass</a>
-  return a low pass filter miir(). A Butterworth filter is used.
   <a href="matlab:help miir/private/parseFilterParams">miir/private/
parseFilterParams</a> -  parses the input plist and returns a full plist for
   <a href="matlab:help miir/private/retrieve">miir/private/retrieve</a>          -
miir objects with given ids from the LTPDA repository specified by the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get a fsdata property.

SYNOPSIS ^

function val = get(fsd, propName)

DESCRIPTION ^

```
 GET get a fsdata property.

%%%
```

SOURCE CODE ^

```
function val = get(fsd, propName)
% GET get a fsdata property.
%
%%%
```

ltpda_ss_modify_time_setup

PURPOSE ^

timeModify is the generic function to decalre or load a subsystem.

SYNOPSIS ^

function varargout = ltpda_ss_modify_time_setup(varargin)

DESCRIPTION ^

```
 timeModify is the generic function to decalre or load a subsystem.
%%%
```

```
 'BMATS' cell array (contains B matrices)
 'CMAT' cell array (contains C matrix)
 'DMATS' cell array (contains D matrices)
 ***** THERE ARE NO DEFAULT PARAMETERS *****

 VERSION: $Id: ltpda_ss_modify_time_setup.m,v 1.3 2008/02/25 12:43:44 adrien Exp $

 HISTORY: 08-02-2008 A Grynagier
 24-01-2008 A Grynagier
 Creation 16-01-2008 A Grynagier

 to do : modify for noise shape filter case or HF inputs with aliasing
%%%
```

```
% 'NAME' is a string
% 'TIMESTEP' is a real positive integer, set to zero in continuous case
% 'PARAMNAMES' cell array of strings describing parameter variables
% 'PARAMVALUES' array of doubles (means expected of parameters)
% 'PARAMSIGMAS' array of doubles (variance expected of parameters)
% 'NBINPUTS' integer number of inputs
% 'INPUTNAMES' cell array of strings (name of the inputs)
% 'INPUTSIZES'array of integers (nb of dimensions of the inputs),
% 'INPUTISUSED' double array (binary to indicate which inputs are used)
% 'AMAT' cell array (contains A matrix)
% 'BMATS' cell array (contains B matrices)
% 'CMAT' cell array (contains C matrix)
% 'DMATS' cell array (contains D matrices)
% ***** THERE ARE NO DEFAULT PARAMETERS *****
%
% VERSION: $Id: ltpda_ss_modify_time_setup.m,v 1.3 2008/02/25 12:43:44 adrien Exp $
%
% HISTORY: 08-02-2008 A Grynagier
% 24-01-2008 A Grynagier
% Creation 16-01-2008 A Grynagier
%
% to do : modify for noise shape filter case or HF inputs with aliasing
%%%
```

```
BMATS = find(System,'BMATS');

%% take different actions depending on old and new timestep
if SystemTimestep == NewTimeStep
    action = 'DoNothing';
elseif NewTimeStep == 0
    action = 'MakeContinuous';
    display('warning because system is sent back to continuous time');
elseif SystemTimestep == 0
    action = 'Discretize';
elseif floor(NewTimeStep/SystemTimestep) == NewTimeStep/SystemTimestep
    action = 'TakeMultiple';
elseif NewTimeStep > SystemTimestep
    action = 'MakeLonger';
else
    action = 'MakeShorter';
    display('warning because system is sent back to shorter time step');
end

%% proceed with matrix modifications
if isequal(action,'DoNothing')
elseif isequal(action,'Discretize')
    A_mat = AMAT{1};
    A_mat_d = expm(A_mat * NewTimeStep);
    NSubstep = 15;
    A_mat_input_d = zeros( size(A_mat) );
    for i = 1:NSubstep
        position = NewTimeStep*(2*i-1)/(NSubstep);
        A_mat_input_d = A_mat_input_d + expm( A_mat*position)/NSubstep;
    end
    AMAT{1} = A_mat_d;
    for i=1:length(BMATS)
        BMATS{i} = A_mat_input_d * BMATS{i};
    end
elseif isequal(action,'TakeMultiple')
    A_mat_1 = AMAT{1};
    multiple = NewTimeStep/SystemTimestep;
    A_mat_2 = A_mat_1^multiple;
    A_mat_input_2 = zeros( size(A_mat_2) );
    for i = 1:multiple
        A_mat_input_2 = A_mat_input_2 + A_mat_2^(i-1);
    end
    AMAT{1} = A_mat_2;
    for i=1:length(BMATS)
        BMATS{i} = A_mat_input_2 * BMATS{i};
    end
elseif isequal(action,'MakeContinuous')
    A_mat_d = AMAT{1};
    [V, E] = eig(A_mat_d);
    A_mat = V * diag(diag(log(E))) * inv(V)/SystemTimestep;
    NSubstep = 15;
    A_mat_input_d = zeros( size(A_mat) );
    for i = 1:NSubstep
```

```
        position = NewTimeStep*(2*i-1)/(NSubstep);
        A_mat_input_d = A_mat_input_d + expm( A_mat*position)/NSubstep;
    end
    A_mat_input_d2c = inv(A_mat_input_d);
    AMAT{1} = A_mat;
    for i=1:length(BMATS)
        BMATS{i} = A_mat_input_d2c * BMATS{i};
    end
elseif isequal(action,'MakeLonger')||isequal(action,'MakeShorter')
    A_mat_1 = AMAT{1};
    [V, E] = eig(A_mat_1);
    A_mat = V * diag(diag(log(E))) * inv(V);
    A_mat_2 = expm( A_mat * NewTimeStep/SystemTimestep);
    NSubstep = 15;
    A_mat_input_1 = zeros( size(A_mat) );
    A_mat_input_2 = zeros( size(A_mat) );
    for i = 1:NSubstep
        position_1 = NewTimeStep*(2*i-1)/(NSubstep);
        position_2 = SystemTimestep*(2*i-1)/(NSubstep);
        A_mat_input_1 = A_mat_input_1 + expm( A_mat*position_1 )/NSubstep;
        A_mat_input_2 = A_mat_input_2 + expm( A_mat*position_2 )/NSubstep;
    end
    AMAT{1} = A_mat_2;
    A_mat_input_2inv = inv(A_mat_input_2);
    for i=1:length(BMATS)
        BMATS{i} = A_mat_input_2 * A_mat_input_2inv * BMATS{i};
    end
end

%% Save Matrix modifications
System = pset(System,'AMAT',AMAT);
System = pset(System,'BMATS',BMATS);
varargout = pset(System,'TIMESTEP',NewTimeStep);
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

set

PURPOSE ^

SET set a cdata property.

SYNOPSIS ^

function c = set(c, varargin)

DESCRIPTION ^

```
 SET set a cdata property.

%%%
```

- cdata CDATA constant data object class constructor.

SOURCE CODE ^

```
function c = set(c, varargin)
% SET set a cdata property.
%
%%%
```

```
%%% Check if this is a special call: default parameter list, cvs-version, category
if ~isempty(varargin{1}) == 1 && ischar(varargin{1})
  in = varargin{1};
  if strcmp(in, 'Params')
    c = DEFAULT_PLIST;
    return
  elseif strcmp(in, 'Version')
    c = VERSION;
    return
  elseif strcmp(in, 'Category')
    c = CATEGORY;
    return
  end
end

if nargin >= 2 && isa(propArgin{1}, 'plist')
  pl         = propArgin{1};
  propArgin = {};
  for ii = 1:nparams(pl)
    %% The 'key' name of a plist/param is always stored with upper case
    %% characters. To set a ao property with this key we have to lower the
    %% 'key' characters.
    propArgin{end+1} = lower(pl.params(ii).key);
    propArgin{end+1} =       pl.params(ii).val;
  end
end

while length(propArgin) >= 2

  prop       = propArgin{1};
  val        = propArgin{2};
  propArgin = propArgin(3:end);

  %%% Special case: If propName is equal to 'vals' then set the 'y' property
  if strcmp(prop, 'vals')
    for jj = 1:numel(c)
      c(jj).y = val;
    end

  %%% Special case: If propName is equal to 'tags' then set the 'x' property
  elseif strcmp(prop, 'tags')
    for jj = 1:numel(c)
      c(jj).x = val;
    end

  %%% Normal case
  elseif ismember(prop, cdata_fields)
    for jj = 1:numel(c)
      c(jj).(prop) = val;
    end
  else
```

```
    error('### ''%s'' is not a valid %s-object property.', prop, class(c));
  end

end

if (nargout == 0)
  display(c);
  error('\n### No output variable!\n### Please use: %s = set(%s, ''key'', ''value'');', inputname(1), inputname(1));
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: param %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: param   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help param/cat">param/cat</a>      -  concatonate params into a
vector.
   <a href="matlab:help param/char">param/char</a>     -  convert a param object
into a string.
   <a href="matlab:help param/display">param/display</a>  -  display a parameter
   <a href="matlab:help param/eq">param/eq</a>       -  overloads the == operator
for param objects.
   <a href="matlab:help param/get">param/get</a>      -  get parameter properties.
   <a href="matlab:help param/isfield">param/isfield</a>  -  tests if the given
field is one of the object properties.
   <a href="matlab:help param/mux">param/mux</a>      -  concatonate params into a
vector.
   <a href="matlab:help param/ne">param/ne</a>       -  overloads the ~= operator
for param objects.
   <a href="matlab:help param/param">param/param</a>    -  Parameter object class
constructor.
   <a href="matlab:help param/save">param/save</a>     -  a param object to file.
   <a href="matlab:help param/set">param/set</a>      -  set a parameter property.
   <a href="matlab:help param/subsasgn">param/subsasgn</a> -  define index
assignment for parameter properties.
   <a href="matlab:help param/subsref">param/subsref</a>  -  Define field name
indexing for parameter objects.
   <a href="matlab:help param/xml">param/xml</a>      -  convert a param object to
an DOM XML object.
   <a href="matlab:help param/xmladd">param/xmladd</a>   -  add a parameter object
to the given node of the input XML
   <a href="matlab:help param/xmlparse">param/xmlparse</a> -  creates a parameter
from a parameter xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get history object properties.

SYNOPSIS ^

function val = get(h, propName)

DESCRIPTION ^

```
 GET get history object properties.

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

 DESCRIPTION: GET get history object properties.

 CALL: name    = get(h, 'name');
       ver     = get(h, 'version');
       pl      = get(h, 'plist');
       inhist  = get(h, 'inhist');
       invars  = get(h, 'invars');
       n       = get(h, 'n');
       pn      = get(h, 'pn');
       created = get(h, 'created');
       consver = get(h, 'consver');

 VERSION: $Id: get.m,v 1.7 2008/02/11 17:28:45 ingo Exp $

 HISTORY: 30-01-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- char CHAR convert a param object into a string.
- 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.

- string STRING writes a command string that can be used to recreate the input history object.

SOURCE CODE ^

```
function val = get(h, propName)
% GET get history object properties.
%
%%%
```

ltpda_ss_pz2ss

PURPOSE ^

% conversion from a pole zero model

SYNOPSIS ^

function varargout = ltpda_ss_pz2ss(varargin)

DESCRIPTION ^

```
% conversion from a pole zero model
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpda_ss_make ltpda_ss_make is the generic function to decalre or load a subsystem.
- utp_ltpda_ss_pz2ss test conversion from state-space to pole zero model

SOURCE CODE ^

```
%% conversion from a pole zero model

function varargout = ltpda_ss_pz2ss(varargin)
% ltpda_pz2ss converts a pole-zero model into a subsystem.
%%%
```

```
%     'NBINPUTS', NBINPUTS ,'INPUTNAMES', INPUTNAMES ,'INPUTSIZES', INPUTSIZES ,'INPUTISUSED', INPUTISUSED ,...
%     'AMAT', AMAT ,'BMATS', BMATS ,'CMAT', CMAT ,'DMATS', DMATS );% PARAMETERS:
% 'TYPE' should be 'SUBSYSTEM'
% 'NAME' is a string
% 'TIMESTEP' is a real positive integer, set to zero in continuous case
% 'PARAMNAMES' cell array of strings describing parameter variables
% 'PARAMVALUES' array of doubles (means expected of parameters)
% 'PARAMSIGMAS' array of doubles (variance expected of parameters)
% 'NBINPUTS' integer number of inputs
% 'INPUTNAMES' cell array of strings (name of the inputs)
% 'INPUTSIZES'array of integers (nb of dimensions of the inputs),
% 'INPUTISUSED' double array (binary to indicate which inputs are used)
% 'AMAT' cell array (contains A matrix)
% 'BMATS' cell array (contains B matrices)
% 'CMAT' cell array (contains C matrix)
% 'DMATS' cell array (contains D matrices)
% ***** THERE ARE NO DEFAULT PARAMETERS *****
%
% VERSION: $Id: ltpda_ss_pz2ss.m,v 1.3 2008/02/25 12:43:44 adrien Exp $
%
% TO DO : check acceptability of deconv function output
% lots of testing including numerical checks.
% Discuss the possiblity of jordan block form and PFD form for the output
% pz model
%
% HISTORY: 01-02-2008 A Grynagier
%%%
```

```
    if isequal(p.name,'real pole')
        a_p = [1 -1/(w0)];
    elseif isequal(p.name,'complex pole')
        q  = get(poles(i), 'q');
        a_p = [1 w0/q w0^2];
    else
        error('type of pole not expected')
    end
    a = conv(a,a_p); %multplication of polynomials to get denominator
end
%% building denominator
Zeros = pzm.zeros;
b = 1;
for i = 1:length(Zeros)
    z = Zeros(i);
    w0 = 2*pi*get(z, 'f');
    if isequal(z.name,'real zero')
        b_z = [1 -w0];
    elseif isequal(z.name,'complex zero')
        q  = get(Zeros(i), 'q');
        b_z = [1 w0/q w0^2];
    else
        error('type of zero not expected')
    end
    b = conv(b,b_z); %multplication of polynomials to get numerator
end

%% building matrices
nss = size(b,2)-1;
[d,a2] = deconv(a,b);
A = [zeros(nss-1, 1) eye(nss-1); fliplr(b(1,2:nss+1))];
B = [zeros(nss-1, 1); 1];
C = fliplr(a2(2:nss+1));
D = d;

%% parsing subsystem
varargout = plist('TYPE', 'SUMBSYSTEM' ,'NAME', pzm.name ,'TIMESTEP', 0 , ...
    'XISOUTPUT', 0,'YISOUTPUT',1,'XINI', zeros(nss,1) , ...
    'PARAMNAMES', {},'PARAMVALUES', [],'PARAMSIGMAS', [],...
    'NBINPUTS', 1 ,'INPUTNAMES', {'U_1'} ,'INPUTSIZES', 1 , 'INPUTISUSED',
1 ,...
    'AMAT', {A} ,'BMATS', {B} ,'CMAT', {C} ,'DMATS', {D} );
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

set

PURPOSE ^

SET set a fsdata property.

SYNOPSIS ^

function fsd = set(fsd, varargin)

DESCRIPTION ^

```
 SET set a fsdata property.

%%%
```

SOURCE CODE ^

```
function fsd = set(fsd, varargin)
% SET set a fsdata property.
%
%%%
```

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: param/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: param/private   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help param/private/fromxml">param/private/fromxml</a>  -  read
from part of an XML tree and create a param object.
   <a href="matlab:help param/private/retrieve">param/private/retrieve</a> -  param
objects with given ids from the LTPDA repository specified by the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get mfir properties.

SYNOPSIS ^

function val = get(varargin)

DESCRIPTION ^

```
 GET get mfir properties.

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

 DESCRIPTION: GET get mfir properties.

 CALL:        get(mfir)
              val     = get(mfir, property);
              name    = get(mfir, 'name');
              fs      = get(mfir, 'fs');
              ntaps   = get(mfir, 'ntaps');
              a       = get(mfir, 'a');
              gd      = get(mfir, 'gd');
              gain    = get(mfir, 'gain');
              history = get(mfir, 'histout');
              infile  = get(mfir, 'infile');
              plist   = get(mfir, 'plist');
              created = get(mfir, 'created');
              version = get(mfir, 'version');

 VERSION:     $Id: get.m,v 1.4 2008/02/11 17:28:45 ingo Exp $

 HISTORY:     30-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- display DISPLAY overloads display functionality for mfir objects.
- mfir MFIR FIR filter object class constructor.
- redesign REDESIGN redesign the input filter to work for the given sample rate.
- string STRING writes a command string that can be used to recreate the input filter object.

SOURCE CODE ^

```
function val = get(varargin)
% GET get mfir properties.
%
%%%
```

ltpda_ss_ss2iir

PURPOSE ^

makeSubsystem is the generic function to decalre or load a subsystem.

SYNOPSIS ^

function varargout = ltpda_ss_ss2iir(varargin)

DESCRIPTION ^

```
 makeSubsystem is the generic function to decalre or load a subsystem.
%%%
```

SOURCE CODE ^

```
function varargout = ltpda_ss_ss2iir(varargin)
% makeSubsystem is the generic function to decalre or load a subsystem.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: ltpda_ss_tofiir
%
% CALL: iir = ltpda_ss_tofiir(varargin)
%
% INPUTS: subsys - subsystem dscribed by a plist
%
% OUTPUTS: iir - a miiir object
%
% ***** THERE ARE NO DEFAULT PARAMETERS *****
%
% VERSION: $Id: ltpda_ss_ss2iir.m,v 1.2 2008/02/25 12:43:44 adrien Exp $
%
% HISTORY: 10-02-2008
% 22-01-2008 A Grynagier
% Creation 02-01-2008 A Grynagier
%
% To do :  more checks! Numerical and syntaxic.  UNDERSTAND WHY CURRENT
% MIIR CALL DOES NOT WORK
% Also discuss the case when the ouput is in PFD form...
% Replace output cell array by output array
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% standard calls for LTPDA function data

ALGONAME = mfilename;
VERSION = '$Id: ltpda_ss_ss2iir.m,v 1.2 2008/02/25 12:43:44 adrien Exp $';
CATEGORY = 'STATESPACE';
display(['starting ' ALGONAME]);

if isequal( varargin{1}, 'Version')
    varargout = VERSION;
    return;
elseif isequal(varargin{1}, 'Params')
    varargout = plist();
    return;
elseif isequal(varargin{1}, 'Category')
    varargout = CATEGORY;
    return;
end

%% proceeding data
syst = varargin{1};

AMAT = find(syst,'AMAT');
CMAT = find(syst,'CMAT');
```

```
BMATS = find(syst,'BMATS');
DMATS = find(syst,'DMATS');
Ninputs = find(syst,'NBINPUTS');
NAME = find(syst,'NAME');
% INPUTSIZES = find(syst,'IMPUTSIZES');
INPUTNAMES = find(syst,'INPUTNAMES');
STEPSIZE = find(syst,'TIMESTEP');

syms s
%found no other way to declare right size of iir array
iirs{1,1,1} = miir();
if STEPSIZE == 0
    %to do instead : discretize system to 0.1s
    display('Error because timestep must be nonzero to make a time discrete filter');
    STEPSIZE = 1;
end
for i=1:Ninputs
    if size(AMAT{1},1)==0
        D = DMATS{i};
        C = CMAT{1};
        for j=1:size(C,1)
            for k=1:size(D,2)
                a = D(j,k);
                b = 1;
                iirs{i,j,k} = miir(plist('a', a, 'b', b, 'gain', 1, 'ntaps', 1, 'name', [NAME,'_',INPUTNAMES{i}], 'fs', 1/STEPSIZE));
            end
        end
    elseif size(BMATS{i},2)==0 ||size(CMAT{1},1)==0
        iirs{i,1,1} = {};
    else
        C = CMAT{1};
        B = BMATS{i};
        D = DMATS{i};
        for j=1:size(C,1)
            for k=1:size(B,2)
                [a,b] = ss2tf(AMAT{1}, B(:,k), C(j,:), D(j,k));
                Gain=1;
                m = miir();
                m.a = a;
                m.b = b;
                m.fs = 1/STEPSIZE;
                m.name = [NAME,'_',INPUTNAMES{i}];
                m.gain = Gain;
                iirs{i,j,k} = m;
%                 iirs{i,j,k} = miir(plist('a', a, 'b', b, 'gain', Gain, 'ntaps', 1, 'name', [NAME,'_',INPUTNAMES{i}], 'fs', 1/STEPSIZE));
            end
        end
    end
end
varargout = iirs;
```

```
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

set

PURPOSE ^

SET set a history property.

SYNOPSIS ^

function h = set(h, varargin)

DESCRIPTION ^

```
 SET set a history property.

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

 DESCRIPTION: SET set a history property.

 CALL: h = set(h, 'name',    'name');
       h = set(h, 'inhist',   history);
       h = set(h, 'invars',   {});
       h = set(h, 'plist',    plist);
       h = set(h, 'n',        n);
       h = set(h, 'pn',       pn);
       h = set(H, 'created',  time());
       h = set(h, 'version', 'history/set.m,v 1.7...');
       h = set(h, 'consver', 'history/set.m,v 1.7...');

 VERSION: $Id: set.m,v 1.11 2008/02/12 15:20:32 ingo Exp $

 HISTORY: 30-01-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- display DISPLAY implement terminal display for history object.

This function is called by:

- plot PLOT plots a history object as a tree diagram.

SOURCE CODE ^

```
function h = set(h, varargin)
% SET set a history property.
%
%%%
```

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: plist %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: plist   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help plist/append">plist/append</a>   -  append a parameter 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/display">plist/display</a>  -  display plist object.
   <a href="matlab:help plist/eq">plist/eq</a>       -  overloads the == operator
for plist objects.
   <a href="matlab:help plist/find">plist/find</a>     -  overloads find routine for
a parameter list.
   <a href="matlab:help plist/get">plist/get</a>      -  get parameter list
properties.
   <a href="matlab:help plist/isfield">plist/isfield</a>  -  tests if the given
field is one of the object properties.
   <a href="matlab:help plist/ne">plist/ne</a>       -  overloads the ~= operator
for param objects.
   <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/pzmresp">plist/pzmresp</a>  -  shadow function for the
pzmodel/resp.
   <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.
   <a href="matlab:help plist/save">plist/save</a>     -  a plist object to file.
   <a href="matlab:help plist/set">plist/set</a>      -  set parameter list
properties.
   <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/subsasgn">plist/subsasgn</a> -  define index
assignment for parameter list properties.
```

```
   <a href="matlab:help plist/subsref">plist/subsref</a>  -  Define field name
indexing for parameter list objects.
   <a href="matlab:help plist/xml">plist/xml</a>      -  convert a plist object to
an DOM XML object.
   <a href="matlab:help plist/xmladd">plist/xmladd</a>   -  add a plist object to
the given node of the input XML DOM
   <a href="matlab:help plist/xmlparse">plist/xmlparse</a> -  creates a parameter
list from a plist xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get miir properties.

SYNOPSIS ^

function val = get(varargin)

DESCRIPTION ^

```
 GET get miir properties.

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

 DESCRIPTION: GET get miir properties.

 CALL:        get(miir)
              val     = get(miir, property);
              name    = get(miir, 'name');
              version = get(miir, 'version');
              created = get(miir, 'created');
              fs      = get(miir, 'fs');
              ntaps   = get(miir, 'ntaps');
              a       = get(miir, 'a');
              b       = get(miir, 'b');
              gain    = get(miir, 'gain');
              infile  = get(miir, 'infile');
              history = get(miir, 'histin');
              history = get(miir, 'histout');
              pl      = get(miir, 'plist');

 VERSION:     $Id: get.m,v 1.6 2008/02/11 17:28:45 ingo Exp $

 HISTORY:     30-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- display DISPLAY overloads display functionality for miir objects.
- miir MIIR IIR filter object class constructor.
- redesign REDESIGN redesign the input filter to work for the given sample rate.
- string STRING writes a command string that can be used to recreate the input filter object.

SOURCE CODE ^

```
function val = get(varargin)
% GET get miir properties.
%
%%%
```

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 1.23 2008/02/21 10:44:09 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 1.23 2008/02/21 10:44:09 hewitson Exp $
%

%------------------------------------------------------------------------
% User Configurable parameters
%

```

```
REPOUSER   = 'hewitson';
REPOPASSWD = '.pw4hewi';

USE_LTPDA_PRINT_SETTINGS = 'Yes';     % Choose LTPDA Print setup
USE_LTPDA_PLOT_SETTINGS  = 'Yes';     % Choose LTPDA Plot setup

REPO_GUI_FONTSIZE   = 10;         % Choose Repository GUI font size
PASSWD_WIN_FONTSIZE = 8;          % Choose login/passw dialog window font size

DBDRIVER = 'com.mysql.jdbc.Driver';   % Set LTPDA Repository database driver

WRAP_STRINGS         = 8;             % Wrap strings at this value
WRAP_LEGEND_STRINGS  = 10;            % Wrap legend strings at this value
XML_SET_SIZE         = 50000;         % Set the maximum line length for writing data sets to XML

TIMEZONE        = 'UTC';                 % Users timezone
                                         % To get valid IDs use:
                                         % java.util.TimeZone.getAvailableIDs
TIME_FORMAT_STR = 'yyyy-mm-dd HH:MM:SS.FFF'; % Users time format.

DEFAULT_DATA_DIR = 'D:\lisa_activity\LTPDA\Analyses';
DEFAULT_ANALYSES_DIR = 'D:\exper_data\LTPDA\data';

%-----------------------------------------------------------------------
%-----------------------------------------------------------------------
% NO NEED TO EDIT BELOW HERE
%-----------------------------------------------------------------------
%-----------------------------------------------------------------------

v = ver('LTPDA');

logo = {...
'                                        ',...
'                  ****                  ',...
'                   **                   ',...
'              -------------             ',...
'          ////       /     \\\\         ',...
'       ///          /          \\\      ',...
'      |            /              |     ',...
'  ** |    +----+  /      +----+    | ** ',...
'  ***|    |    |//-------|    |    |*** ',...
'  ** |    +----+        /+----+    | ** ',...
'     |                /            |     ',...
'       \\\            /        ///      ',...
'          \\\\      //     ////         ',...
'              -------------             ',...
'                   **                   ',...
'                  ****                  ',...
};

l1 = '+-----------------------------------------------------+';
ll = length(l1);
```

```

disp(l1);
disp('|                                                    |')
for j=1:length(logo)
  disp([strpad(sprintf('|      %s  ', char(logo{j})), ll-1) '|']);
end
disp([strpad('|', ll-1) '|'])
disp([strpad(sprintf('|          Welcome to %s', v.Name), ll-1) '|'])
disp([strpad('|', ll-1) '|'])
disp([strpad(sprintf('|                 Version: %s', v.Version), ll-1) '|'])
disp([strpad(sprintf('|                 Release: %s', v.Release), ll-1) '|'])
disp([strpad(sprintf('|                    Date: %s', v.Date), ll-1) '|'])
disp([strpad('|', ll-1) '|'])
disp(l1);

%-----------------------------------------------------------------------
% set page properties for printing
if strcmp(upper(USE_LTPDA_PRINT_SETTINGS), 'YES')
  set(0,'DefaultFigurePaperOrientation','portrait');
  set(0,'DefaultFigurePaperType','A4');
  set(0,'DefaultFigurePaperUnits','inches');
end

%-----------------------------------------------------------------------
% Plot settings
if strcmp(upper(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
setappdata(0, 'ltpda_repo_user', REPOUSER);
setappdata(0, 'ltpda_repo_pass', REPOPASSWD);
setappdata(0, 'mysql_driver', DBDRIVER);

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')) '/../mysql/mysql-connector-
java-5.0.6-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 '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_version', ltpda_version);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

set

PURPOSE ^

SET set a mfir object property.

SYNOPSIS ^

function f = set(f, varargin)

DESCRIPTION ^

```
 SET set a mfir object property.

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

 DESCRIPTION: SET set an mfir object property.

 CALL:        f    = set(f, property, value);
              mfir = set(mfir, 'name',    'name');
              mfir = set(mfir, 'fs',       0);
              mfir = set(mfir, 'ntaps',    0);
              mfir = set(mfir, 'a',        []);
              mfir = set(mfir, 'gd',       0);
              mfir = set(mfir, 'gain',        0);
              mfir = set(mfir, 'histout',  history());
              mfir = set(mfir, 'infile',   'name');
              mfir = set(mfir, 'plist',    plist);
              mfir = set(mfir, 'created',  time());
              mfir = set(mfir, 'version', 'version');

 VERSION:     $Id: set.m,v 1.7 2008/02/13 18:33:19 ingo Exp $

 HISTORY:     30-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- display DISPLAY overloads display functionality for mfir objects.

This function is called by:

- mfir MFIR FIR filter object class constructor.
- redesign REDESIGN redesign the input filter to work for the given sample rate.
- resp RESP Make a frequency response of the filter.

SOURCE CODE ^

```
function f = set(f, varargin)
% SET set a mfir object property.
%
%%%
```

```
0045   display(f);
0046   error('\n### No output variable!\n### Please use: %s = set(%s, ''key'', ''value'');', inputname(1), inputname(1));
0047 end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: plist/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: plist/private   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help plist/private/fromxml">plist/private/fromxml</a>  -  read
from part of an XML tree and create a plist object.
   <a href="matlab:help plist/private/retrieve">plist/private/retrieve</a> -  plist
objects with given ids from the LTPDA repository specified by the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get parameter properties.

SYNOPSIS ^

function val = get(p, propName)

DESCRIPTION ^

```
 GET get parameter properties.

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

 DESCRIPTION: GET get parameter properties.

 CALL:        name    = get(parameter, 'name');
              key     = get(parameter, 'key');
              val     = get(parameter, 'val');
              created = get(parameter, 'created');
              version = get(parameter, 'version');

 VERSION:     $Id: get.m,v 1.5 2008/02/11 17:28:45 ingo Exp $

 HISTORY:     30-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- char CHAR convert a param object into a string.
- display DISPLAY display a parameter

SOURCE CODE ^

```
function val = get(p, propName)
% GET get parameter properties.
```

```
%
%%%
```

ltpda_str2cells

PURPOSE ^

LTPDA_STR2CELLS Take a single string and separate out individual "elements" into a new cell array.

SYNOPSIS ^

function newCell = ltpda_str2cells(someString)

DESCRIPTION ^

```
 LTPDA_STR2CELLS Take a single string and separate out individual "elements" into a
new cell array.

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

 DESCRIPTION: LTPDA_STR2CELLS Take a single string and separate out individual
              "elements" into a new cell array. Elements are defined as non-blank
characters separated by
              spaces.

              Similar to str2cell, except str2cell requires an array of strings.
              str2cells requires only 1 string.

 CALL:       newCell = ltpda_str2cells(aString)

 INPUTS:     aString - string

 OUTPUTS:    newCell - cell array of strings

 EXAMPLE:    Consider the following string in the workspace:

 aString = ' a  b        c  d  efgh ij      klmnopqrs t u v w xyz    '
 newCell = 'a'
           'b'
           'c'
           'd'
           'efgh'
           'ij'
           'klmnopqrs'
           't'
           'u'
           'v'
           'w'
           'xyz'
```

```
 REMARK:  This is copied from a file found on MathWorks File Exchange.

 VERSION: $Id: ltpda_str2cells.m,v 1.7 2007/07/13 12:17:39 ingo Exp $

 HISTORY: 26-01-2007 M Hewitson
             Creation

%%%
```

```
%           'u'
%           'v'
%           'w'
%           'xyz'
%
% REMARK:  This is copied from a file found on MathWorks File Exchange.
%
% VERSION: $Id: ltpda_str2cells.m,v 1.7 2007/07/13 12:17:39 ingo Exp $
%
% HISTORY: 26-01-2007 M Hewitson
%             Creation
%
%%%
```

set

PURPOSE ^

SET set a miir object property.

SYNOPSIS ^

function f = set(f, varargin)

DESCRIPTION ^

```
 SET set a miir object property.

%%%
```

This function is called by:

- miir MIIR IIR filter object class constructor.
- redesign REDESIGN redesign the input filter to work for the given sample rate.
- resp RESP Make a frequency response of the filter.

SOURCE CODE ^

```
function f = set(f, varargin)
% SET set a miir object property.
%
%%%
```

```
f = generic_set(f, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

if (nargout == 0)
  display(f);
  error('\n### No output variable!\n### Please use: %s = set(%s, ''key'', ''value'');', inputname(1), inputname(1));
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: pole %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: pole   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help pole/cat">pole/cat</a>      -  concatonate poles into a
vector.
   <a href="matlab:help pole/char">pole/char</a>     -  convert a pole object into a
string.
   <a href="matlab:help pole/cp2iir">pole/cp2iir</a>   -  Return a,b IIR filter
coefficients for a complex pole designed using the bilinear transform.
   <a href="matlab:help pole/display">pole/display</a>  -  overloads display
functionality for pole objects.
   <a href="matlab:help pole/eq">pole/eq</a>       -  overloads the == operator for
pole objects.
   <a href="matlab:help pole/get">pole/get</a>      -  get pole properties.
   <a href="matlab:help pole/isfield">pole/isfield</a>  -  tests if the given field
is one of the object properties.
   <a href="matlab:help pole/ne">pole/ne</a>       -  overloads the ~= operator for
pole objects.
   <a href="matlab:help pole/pole">pole/pole</a>     -  construct a pole object.
   <a href="matlab:help pole/resp">pole/resp</a>     -  returns the complex response
of the pole object.
   <a href="matlab:help pole/rp2iir">pole/rp2iir</a>   -  Return a,b coefficients
for a real pole designed using the bilinear transform.
   <a href="matlab:help pole/save">pole/save</a>     -  a pole object to file.
   <a href="matlab:help pole/set">pole/set</a>      -  set a pole property.
   <a href="matlab:help pole/string">pole/string</a>   -  writes a command string
that can be used to recreate the input pole object.
   <a href="matlab:help pole/subsref">pole/subsref</a>  -  Define field name
indexing for pole objects.
   <a href="matlab:help pole/xml">pole/xml</a>      -  convert a pole object to an
DOM XML object.
   <a href="matlab:help pole/xmladd">pole/xmladd</a>   -  add a pole object to the
given node of the input XML DOM
   <a href="matlab:help pole/xmlparse">pole/xmlparse</a> -  creates a pole from a
pole xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_strs2cells

PURPOSE ^

LTPDA_STRS2CELLS convert a set of input strings to a cell array.

SYNOPSIS ^

function cell = ltpda_strs2cells(varargin)

DESCRIPTION ^

```
 LTPDA_STRS2CELLS convert a set of input strings to a cell array.

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

 DESCRIPTION: LTPDA_STRS2CELLS convert a set of input strings to a cell array.

 CALL:       cell = ltpda_strs2cells(str1, str2, ...)

 INPUTS:     str1 - string
             str2 - string
                     ...

 OUTPUTS:    cell - cell array of the input strings

 VERSION: $Id: ltpda_strs2cells.m,v 1.5 2007/07/13 12:17:39 ingo Exp $

 HISTORY: 26-01-2007 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function cell = ltpda_strs2cells(varargin)
% LTPDA_STRS2CELLS convert a set of input strings to a cell array.
%
```

```
%%%
```

set

PURPOSE ^

SET set a parameter property.

SYNOPSIS ^

function p = set(p, varargin)

DESCRIPTION ^

```
 SET set a parameter property.

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

 DESCRIPTION: SET set a parameter property.

 CALL: p = set(p, 'key',     'name');
       p = set(p, 'val',      val);
       p = set(p, 'val',     'val');
       p = set(p, 'name',    'my name');
       p = set(p, 'created',  time-object);

 VERSION: $Id: set.m,v 1.9 2008/02/12 15:20:32 ingo Exp $

 HISTORY: 30-01-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- display DISPLAY display a parameter

This function is called by:

SOURCE CODE ^

```
function p = set(p, varargin)
% SET set a parameter property.
%
```

```
%%%
```

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: pole/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%   class: pole/private   %%%%%%%%%%%%%%%%%%

   <a href="matlab:help pole/private/fromxml">pole/private/fromxml</a>  -  read from
part of an XML tree and create a zero object.
   <a href="matlab:help pole/private/pfq2ri">pole/private/pfq2ri</a>   - Convert
frequency/Q pole/zero representation into real
   <a href="matlab:help pole/private/pri2fq">pole/private/pri2fq</a>   - Convert
comlpex pole/zero into frequency/Q pole/zero representation.
   <a href="matlab:help pole/private/retrieve">pole/private/retrieve</a> -  pole
objects with given ids from the LTPDA repository specified by the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get pole properties.

SYNOPSIS ^

function val = get(p, propName)

DESCRIPTION ^

```
 GET get pole properties.

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

 DESCRIPTION: GET get pole properties.

 CALL:        name    = get(p, 'name');
              f       = get(p, 'f');
              q       = get(p, 'q');
              ri      = get(p, 'ri');
              created = get(p, 'created');
              version = get(p, 'version');

 VERSION:     $Id: get.m,v 1.6 2008/02/11 17:28:45 ingo Exp $

 HISTORY:     02-04-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- cp2iir CP2IIR Return a,b IIR filter coefficients for a complex pole designed using the bilinear transform.
- display DISPLAY overloads display functionality for pole objects.
- resp RESP returns the complex response of the pole object.
- rp2iir RP2IIR Return a,b coefficients for a real pole designed using the bilinear transform.

SOURCE CODE ^

```
function val = get(p, propName)
% GET get pole properties.
%
%%%
```

ltpda_structcat

PURPOSE ^

LTPDA_STRUCTCAT concatonate structures to make one large structure.

SYNOPSIS ^

function out = ltpda_structcat(varargin)

DESCRIPTION ^

```
 LTPDA_STRUCTCAT concatonate structures to make one large structure.

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

 DESCRIPTION: LTPDA_STRUCTCAT concatonate structures to make one large
              structure.

 CALL:        out = ltpda_structcat(struct1, struct2, ...)

 INPUTS:      struct1 - a structure
              struct2 - a structure

 OUTPUTS:     out - structure with all fields of input structures.

 VERSION: $Id: ltpda_structcat.m,v 1.5 2007/07/13 12:17:39 ingo Exp $

 HISTORY: 26-01-2007 M Hewitson
              Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function out = ltpda_structcat(varargin)
% LTPDA_STRUCTCAT concatonate structures to make one large structure.
%
```

```
%%%
```

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/display">provenance/display</a>    -  overload
terminal display for provenance objects.
   <a href="matlab:help provenance/eq">provenance/eq</a>         -  overloads the ==
operator for provenance objects.
   <a href="matlab:help provenance/get">provenance/get</a>        -  get provenance
properties.
   <a href="matlab:help provenance/isfield">provenance/isfield</a>    -  tests if
the given field is one of the object properties.
   <a href="matlab:help provenance/ne">provenance/ne</a>         -  overloads the ~=
operator for provenance objects.
   <a href="matlab:help provenance/provenance">provenance/provenance</a> -
constructors for provenance class.
   <a href="matlab:help provenance/save">provenance/save</a>       -  a provenance
object to file.
   <a href="matlab:help provenance/set">provenance/set</a>        -  set an
provenance property.
   <a href="matlab:help provenance/subsref">provenance/subsref</a>    -  Define
field name indexing for provenance objects.
   <a href="matlab:help provenance/xml">provenance/xml</a>        -  convert a
provenance object to an DOM XML object.
   <a href="matlab:help provenance/xmladd">provenance/xmladd</a>     -  adds specwin
object to the given node of the xml input
   <a href="matlab:help provenance/xmlparse">provenance/xmlparse</a>   -  creates a
provenance from a provenance xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get provenance properties.

SYNOPSIS ^

function val = get(p, propName)

DESCRIPTION ^

```
 GET get provenance properties.

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

 DESCRIPTION: GET get provenance properties.

 CALL:        creator     = get(p, 'creator');
              created     = get(p, 'created');
              ip          = get(p, 'ip');
              hostname    = get(p, 'hostname');
              os          = get(p, 'os');
              matlab_ver  = get(p, 'matlab_version');
              sigproc_ver = get(p, 'sigproc_version');
              ltpda_ver   = get(p, 'ltpda_version');

 VERSION:     $Id: get.m,v 1.4 2008/02/11 17:28:45 ingo Exp $

 HISTORY:     30-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- char CHAR convert a provenance object into a string.
- display DISPLAY overload terminal display for provenance objects.

SOURCE CODE ^

```
function val = get(p, propName)
% GET get provenance properties.
%
%%%
```

ltpda_structidx

PURPOSE ^

LTPDA_STRUCTIDX Index a structure.

SYNOPSIS ^

function sout = ltpda_structidx(sin, idx, varargin)

DESCRIPTION ^

```
 LTPDA_STRUCTIDX Index a structure.

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

 DESCRIPTION: LTPDA_STRUCTIDX Index a structure. All fields are assumed
              to be vectors of the same length, except those explicitely ignored.

             You can give an additional list of field names which will be
             ignored in the indexing.

 CALL:       sout = ltpda_structidx(sin, idx)
                  = ltpda_structidx(sin, idx, 'field1', 'field2')

 INPUTS:     sin    - a struct
             idx    - index
             field1 - field name
             field2 - field name
                     ...

 OUTPUTS:    sout   - a struct

 VERSION: $Id: ltpda_structidx.m,v 1.5 2007/07/13 12:17:39 ingo Exp $

 HISTORY: 26-01-2007 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function sout = ltpda_structidx(sin, idx, varargin)
% LTPDA_STRUCTIDX Index a structure.
%
%%%
```

```
  if ~skip
    str = sprintf('sout.%s = sin.%s(idx);', fn, fn);
    eval(str);
  end
end



% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

set

PURPOSE ^

SET set a pole property.

SYNOPSIS ^

function p = set(p, varargin)

DESCRIPTION ^

```
 SET set a pole property.

%%%
```

```
function p = set(p, varargin)
% SET set a pole property.
%
%%%
```

```
  elseif strcmp(in, 'Category')
    p = CATEGORY;
    return
  end
end

if nargin >= 2 && isa(propArgin{1}, 'plist')
  pl         = propArgin{1};
  propArgin = {};
  for ii = 1:nparams(pl)
    %% The 'key' name of a plist/param is always stored with upper case
    %% characters. To set a ao property with this key we have to lower the
    %% 'key' characters.
    propArgin{end+1} = lower(pl.params(ii).key);
    propArgin{end+1} =       pl.params(ii).val;
  end
end

while length(propArgin) >= 2

  prop       = propArgin{1};
  val        = propArgin{2};
  propArgin = propArgin(3:end);

  %%% Special case: If propName is equal to 'f' then we have to compute 'ri'
  if strcmp(prop, 'f')
    for jj = 1:numel(p)
      p(jj).f  = val;
      p(jj).ri = pfq2ri(p(jj).f, p(jj).q);
    end

  %%% Special case: If propName is equal to 'q' then we have to compute 'ri'
  elseif strcmp(prop, 'q')
    for jj = 1:numel(p)
      p(jj).q  = val;
      p(jj).ri = pfq2ri(p(jj).f, p(jj).q);
    end

  %%% Special case: If propName is equal to 'ri' then we have to compute 'q' and 'f'
  elseif strcmp(prop, 'ri')
    if size(val,2) > size(val,1), val = val.'; end
    for jj = 1:numel(p)
      if length(val) == 1
        p(jj).ri = [val; conj(val)];
      else
        p(jj).ri = val;
      end
      [p(jj).f, p(jj).q] = pri2fq(p(jj).ri);
    end

  %%% Normal case
```

```
  elseif ismember(prop, pole_fields)
    for jj = 1:numel(p)
      p(jj).(prop) = val;
    end
  else
    error('### ''%s'' is not a valid %s-object property.', prop, class(p));
  end

end

if (nargout == 0)
  display(p);
  error('\n### No output variable!\n### Please use: %s = set(%s, ''key'', ''value'');', inputname(1), inputname(1));
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: provenance/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: provenance/private   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help provenance/private/fromxml">provenance/private/fromxml</a>
-  read from part of an XML tree and create a provenance object.
   <a href="matlab:help provenance/private/retrieve">provenance/private/retrieve</a>
-  provenance objects with given ids from the LTPDA repository specified by the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get a pole/zero model property.

SYNOPSIS ^

function val = get(pzm, propName)

DESCRIPTION ^

```
 GET get a pole/zero model property.

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

 DESCRIPTION: GET get a pole/zero model property.

 CALL:        name    = get(pzm, 'name');
              gain    = get(pzm, 'gain');
              poles   = get(pzm, 'poles');
              zeros   = get(pzm, 'zeros');
              created = get(pzm, 'created');
              version = get(pzm, 'version');

 VERSION:     $Id: get.m,v 1.4 2008/02/11 17:28:45 ingo Exp $

 HISTORY:     03-04-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- display DISPLAY overloads display functionality for pzmodel objects.
- 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.

SOURCE CODE ^

```
function val = get(pzm, propName)
% GET get a pole/zero model property.
%
%%%
```

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]
             'Nolap' - segment overlap (number of samples) [default: taken from
window function]
             'Nfft'  - number of samples in each fft [default: fs of input data]

 VERSION:     $Id: ltpda_tfe.m,v 1.14 2007/07/16 12:52:21 ingo 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')
```

```
%%%
```

```
%
% >> pl = ltpda_tfe('Params')
%
%%%
```

```
plo = append(plo, param('Win', specwin('Kaiser', 1, 100)));
plo = append(plo, param('Nolap', -1));
disp('* done.');

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

set

PURPOSE ^

SET set an provenance property.

SYNOPSIS ^

function p = set(p, varargin)

DESCRIPTION ^

```
 SET set an provenance property.

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

 DESCRIPTION: SET set an provenance property.

 CALL:        prov = set(prov, 'creator',        'unknown');
              prov = set(prov, 'created',         time());
              prov = set(prov, 'ip',             '127.0.0.2');
              prov = set(prov, 'hostname',       'hws165');
              prov = set(prov, 'os',             'GLNX86');
              prov = set(prov, 'matlab_version', '7.4 (R2007a)');
              prov = set(prov, 'matlab_version', '6.7 (R2007a)');
              prov = set(prov, 'matlab_version', '0.4 (R2007a)');

 VERSION:     $Id: set.m,v 1.5 2008/02/12 15:20:33 ingo Exp $

 HISTORY:     30-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- display DISPLAY overload terminal display for provenance objects.

This function is called by:

SOURCE CODE ^

```
function p = set(p, varargin)
% SET set an provenance property.
%
%%%
```

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: pzmodel %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%   class: pzmodel   %%%%%%%%%%%%%%%%%%

   <a href="matlab:help pzmodel/char">pzmodel/char</a>         -  convert a pole
object into a string.
   <a href="matlab:help pzmodel/display">pzmodel/display</a>      -  overloads
display functionality for pzmodel objects.
   <a href="matlab:help pzmodel/eq">pzmodel/eq</a>           -  overloads the ==
operator for pzmodel objects.
   <a href="matlab:help pzmodel/get">pzmodel/get</a>          -  get a pole/zero
model property.
   <a href="matlab:help pzmodel/getlowerFreq">pzmodel/getlowerFreq</a> -  gets the
frequency of the lowest pole or zero in the model.
   <a href="matlab:help pzmodel/getupperFreq">pzmodel/getupperFreq</a> -  gets the
frequency of the highest pole or zero in the model.
   <a href="matlab:help pzmodel/isfield">pzmodel/isfield</a>      -  tests if the
given field is one of the object properties.
   <a href="matlab:help pzmodel/ne">pzmodel/ne</a>           -  overloads the ~=
operator for pzmodel objects.
   <a href="matlab:help pzmodel/pzmodel">pzmodel/pzmodel</a>      -  constructor for
pzmodel class.
   <a href="matlab:help pzmodel/resp">pzmodel/resp</a>         -  returns the
complex response of a pzmodel as an Analysis Object.
   <a href="matlab:help pzmodel/save">pzmodel/save</a>         -  a pzmodel object
to file.
   <a href="matlab:help pzmodel/set">pzmodel/set</a>          -  sets a pole/zero
model property.
   <a href="matlab:help pzmodel/string">pzmodel/string</a>       -  writes a command
string that can be used to recreate the input pzmodel object.
   <a href="matlab:help pzmodel/subsref">pzmodel/subsref</a>      -  Define field
name indexing for pole/zero model objects.
   <a href="matlab:help pzmodel/tomfir">pzmodel/tomfir</a>       -  approximates a
pole/zero model with an FIR filter.
   <a href="matlab:help pzmodel/tomiir">pzmodel/tomiir</a>       -  converts a
pzmodel to an IIR filter using a bilinear transform.
   <a href="matlab:help pzmodel/xml">pzmodel/xml</a>          -  convert a pzmodel
```

```
object to an DOM XML object.
   <a href="matlab:help pzmodel/xmladd">pzmodel/xmladd</a>       -  add a pzmodel
object to the given node of the input XML DOM
   <a href="matlab:help pzmodel/xmlparse">pzmodel/xmlparse</a>     -  creates a
pzmodel from a pzmodel xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get specwin object properties.

SYNOPSIS ^

function val = get(w, propName)

DESCRIPTION ^

```
 GET get specwin object properties.

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

 DESCRIPTION: GET get specwin object properties.

 CALL:        name     = get(sw, 'name');
              alpha    = get(sw, 'alpha');
              psll     = get(sw, 'psll');
              rov      = get(sw, 'rov');
              nenbw    = get(sw, 'nenbw');
              w3db     = get(sw, 'w3db');
              flatness = get(sw, 'flatness');
              ws       = get(sw, 'ws');
              ws2      = get(sw, 'ws2');
              win      = get(sw, 'win');
              created  = get(sw, 'created');
              created  = get(sw, 'version');

 VERSION:     $Id: get.m,v 1.4 2008/02/11 17:28:45 ingo Exp $

 HISTORY:     30-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- display DISPLAY overloads display functionality for specwin objects.

SOURCE CODE ^

```
function val = get(w, propName)
% GET get specwin object properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: GET get specwin object properties.
%
% CALL:        name     = get(sw, 'name');
%              alpha    = get(sw, 'alpha');
%              psll     = get(sw, 'psll');
%              rov      = get(sw, 'rov');
%              nenbw    = get(sw, 'nenbw');
%              w3db     = get(sw, 'w3db');
%              flatness = get(sw, 'flatness');
%              ws       = get(sw, 'ws');
%              ws2      = get(sw, 'ws2');
%              win      = get(sw, 'win');
%              created  = get(sw, 'created');
%              created  = get(sw, 'version');
%
% VERSION:     $Id: get.m,v 1.4 2008/02/11 17:28:45 ingo Exp $
%
% HISTORY:     30-01-2007 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION       = '$Id: get.m,v 1.4 2008/02/11 17:28:45 ingo Exp $';
CATEGORY      = 'Helper';
DEFAULT_PLIST =  plist('property', '');

val = generic_get(w, propName, DEFAULT_PLIST, VERSION, CATEGORY);
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_timedomainfit

PURPOSE ^

LTPDA_TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to a target time-series AO.

SYNOPSIS ^

function varargout = ltpda_timedomainfit(varargin)

DESCRIPTION ^

```
 LTPDA_TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to a target time-
series AO.

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

 DESCRIPTION: LTPDA_TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to
              a target time-series AO.

 CALL:        b = ltpda_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


 PARAMETERS:  no valid parameters at the moment.

 OUTPUTS:     b - cdata type analysis object containing the fitted coefficients

 GUI CALL:    The following call returns a parameter list object that contains
              the default parameter values:

              >> pl = ltpda_xcorr('Params')

 VERSION:     $Id: ltpda_timedomainfit.m,v 1.5 2007/11/08 10:35:04 hewitson Exp $

 HISTORY:     01-03-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = ltpda_timedomainfit(varargin)
% LTPDA_TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to a target time-series AO.
%
%%%
```

```
  if strcmp(in, 'Params')
    varargout{1} = getDefaultPL();
    return
  end
end

% capture input variable names
invars = {};
for j=1:nargin
  iname = inputname(j);
  if isempty(iname) & isnumeric(varargin{j})
    iname = num2str(varargin{j});
  elseif isempty(iname) & ischar(varargin{j})
    iname = varargin{j};
  end
  invars = [invars cellstr(iname)];
end

ALGONAME = mfilename;
VERSION  = '$Id: ltpda_timedomainfit.m,v 1.5 2007/11/08 10:35:04 hewitson Exp $';

% Look at inputs
if nargin < 1
  error('### Incorrect number of inputs.')
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('### skipping fsdata/AO.');
      end
    end
  end
end

% unpack parameter list
plo = plist();

% Initialise output
x  = [];
% Loop over analysis objects and build Matrix for fitting
```

```
naFit = length(as);
if naFit < 2
  error('### Please input at least two AOs.');
end

ho = [];
for i=1:naFit

  % get data out
  a  = as(i);
  ho = [ho a.hist];
  d  = a.data;
  dinfo = whos('d');
  if ~isreal(d.y)
    error('### I only work with real time-series at the moment.');
  end
  x(:,i) = d.y;
end
% Fit
[m,n] = size(x);
[coef, stdx, mse, S] = lscov(x(:,2:n),x(:,1));

disp(['*** got coefficients: ' num2str(coef.')]);

% Return new cdata AO with  the coefficients in
% - we add '1' to the front of the coefficients so that ltpda_lincom
%   will work directly.

% make a new history object
h = history(ALGONAME, VERSION, [], ho);
h = set(h, 'invars', invars);

% make output analysis object
bo = ao([1 -coef.']);
bo = setnh(bo, 'name', ['fit of [' sprintf('%s ', char(invars{2:end})) '] to '
char(invars{1})]);
bo = setnh(bo, 'hist', h);

varargout{1} = bo;

%-----------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

disp('* creating default plist...');
plo = plist();
disp('* done.');


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: pzmodel/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: pzmodel/private   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help pzmodel/private/abcascade">pzmodel/private/abcascade</a> -
two filters together to get a new filter.
   <a href="matlab:help pzmodel/private/fromxml">pzmodel/private/fromxml</a>   -
read from part of an XML tree and create a pzmodel object.
   <a href="matlab:help pzmodel/private/pzm2ab">pzmodel/private/pzm2ab</a>    -
convert pzmodel to IIR filter coefficients using bilinear
   <a href="matlab:help pzmodel/private/retrieve">pzmodel/private/retrieve</a>  -
pzmodel objects with given ids from the LTPDA repository specified by the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get time properties.

SYNOPSIS ^

function val = get(tt, prop_name)

DESCRIPTION ^

```
 GET get time properties.

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

 DESCRIPTION: GET get time properties.

 CALL:        value = get(tt, 'property');

 EXAMPLES:    name             = get(tt, 'name');
              utc_epoch_milli  = get(tt, 'utc_epoch_milli');
              timezone         = get(tt, 'timezone');
              timezone_IDs     = get(tt, 'timezone_IDs');
              timezone_IDs     = get(tt, 'IDs');
              timeformat       = get(tt, 'timeformat');
              time_str         = get(tt, 'time_str');
              created          = get(tt, 'created');
              version          = get(tt, 'version');

 VERSION:     $Id: get.m,v 1.10 2008/02/11 17:28:45 ingo Exp $

 HISTORY:     23-07-2007 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a time object into a string.

This function is called by:

SOURCE CODE ^

```
function val = get(tt, prop_name)
% GET get time properties.
%
%%%
```

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:

- explore_ao EXPLORE_AO 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.
%
%   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 $
%

%   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;

if ~isempty(expfcn)
    set(tree, 'NodeExpandedCallback', {@nodeExpanded, tree, expfcn});
end

if ~isempty(selfcn)
    set(tree, 'NodeSelectedCallback', {@nodeSelected, tree, selfcn});
end

end

%% -----------------------------------------------------
function nodeExpanded(src, evd, tree, expfcn)                          %#ok

% tree = handle(src);
% evdsrc = evd.getSource;

evdnode  = evd.getCurrentNode;
% indices = [];

if ~tree.isLoaded(evdnode)
    value = evdnode.getValue;

    % <call a user function(value) which returns uitreenodes>;
    cbk = expfcn;
    if iscell(cbk)
        childnodes = feval(cbk{1}, tree, value, cbk{2:end});
    else
        childnodes = feval(cbk, tree, value);
    end

    if (length(childnodes) == 1)
        % Then we dont have an array of nodes. Create an array.
        chnodes = childnodes;
        childnodes = javaArray('com.mathworks.hg.peer.UITreeNode', 1);
        childnodes(1) = java(chnodes);
    end

```

```
    tree.add(evdnode, childnodes);
    tree.setLoaded(evdnode, true);
end

end

%% -------------------------------------------------------
function nodeSelected(src, evd, tree, selfcn)                          %#ok
cbk = selfcn;
hgfeval(cbk, tree, evd);

end

%% -------------------------------------------------------
function [node, expfcn] = processNode(root)
expfcn = [];

if isempty(root) || isa(root, 'com.mathworks.hg.peer.UITreeNode') || ...
        isa(root, 'javahandle.com.mathworks.hg.peer.UITreeNode')
    node = root;
elseif ishghandle(root)
    % Try to process as an HG object.
    try
        node = uitreenode(handle(root), get(root, 'Type'), ...
            [], isempty(get(0, 'children')));
    catch
        node = [];
    end
    expfcn = @hgBrowser;
elseif ismodel(root)
    % Try to process as an open Simulink system

    % TODO if there is an open simulink system and a directory on the path with
    % the same name, the system will hide the directory. Perhaps we should
    % warn about this.
    try
        h = handle(get_param(root,'Handle'));
        % TODO we pass root to the tree as a string,
        % it would be better if we could just pass the
        % handle up
        node = uitreenode(root, get(h, 'Name'), ...
            [], isempty(h.getHierarchicalChildren));
    catch
        node = [];
    end
    expfcn = @mdlBrowser;
elseif ischar(root)
    % 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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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/display">specwin/display</a>  -  overloads display
functionality for specwin objects.
   <a href="matlab:help specwin/eq">specwin/eq</a>       -  overloads the ==
operator for specfin objects.
   <a href="matlab:help specwin/get">specwin/get</a>      -  get specwin object
properties.
   <a href="matlab:help specwin/isfield">specwin/isfield</a>  -  tests if the given
field is one of the object properties.
   <a href="matlab:help specwin/ne">specwin/ne</a>       -  overloads the ~=
operator for specwin objects.
   <a href="matlab:help specwin/plot">specwin/plot</a>     -  plots a specwin object.
   <a href="matlab:help specwin/save">specwin/save</a>     -  a specwin object to
file.
   <a href="matlab:help specwin/set">specwin/set</a>      -  sets a specwin property.
   <a href="matlab:help specwin/specwin">specwin/specwin</a>  -  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/subsasgn">specwin/subsasgn</a> -  define index
assignment for specwin objects.
   <a href="matlab:help specwin/subsref">specwin/subsref</a>  -  Define field name
indexing for specwin objects.
   <a href="matlab:help specwin/xml">specwin/xml</a>      -  convert a specwin
object to an DOM XML object.
   <a href="matlab:help specwin/xmladd">specwin/xmladd</a>   -  adds specwin object
to the given node of the xml input
   <a href="matlab:help specwin/xmlparse">specwin/xmlparse</a> -  creates a specwin
from a specwin xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get timeformat properties.

SYNOPSIS ^

function val = get(tf, prop_name)

DESCRIPTION ^

```
 GET get timeformat properties.

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

 DESCRIPTION: GET get timeformat properties.

 CALL:        value = get(tf, 'property');

 EXAMPLES:    format_str         = get(tf, 'format_str');
              format_nr          = get(tf, 'format_nr');
              default_matlab_str = get(tf, 'default_matlab_str');
              default_matlab_nr  = get(tf, 'default_matlab_nr');
              default_java_str   = get(tf, 'default_java_str');

 VERSION:     $Id: get.m,v 1.1 2008/02/11 17:28:45 ingo Exp $

 HISTORY:     11-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function val = get(tf, prop_name)
% GET get timeformat properties.
%
```

```
%%%
```

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:

- explore_ao EXPLORE_AO 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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

set

PURPOSE ^

SET set time properties.

SYNOPSIS ^

function tt = set(tt, varargin)

DESCRIPTION ^

```
 SET set time properties.

%%%
```

- set SET set time properties.
- time TIME time object class constructor.

This function is called by:

- format FORMAT Returns the time in the handover format.
- minus MINUS overloads - operator for time objects.
- plus PLUS overloads - operator for time objects.
- set SET set time properties.
- time TIME time object class constructor.

SUBFUNCTIONS ^

- function ret = parse_time_string(in_str)
- function default_plist = GET_DEFAULT_PLIST

SOURCE CODE ^

```
function tt = set(tt, varargin)
% SET set time properties.
%
%%%
```

```
CATEGORY      = 'Helper';

time_fields    = fieldnames(tt);
property_argin = varargin;

%%%
```

```
    % Set the timeformat AND transform the current time string to this format
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (strcmp(prop, 'timeformat')) && (compute)

      if isa(val, 'timeformat')
        tt.timeformat = val;

      else
        t_format = java.text.SimpleDateFormat(tt.timeformat.default_java_str);
        t_format.setTimeZone(tt.timezone);
        current_time_str = char(t_format.format (tt.utc_epoch_milli));

        d_num = datenum (current_time_str, tt.timeformat.default_matlab_str);
        tt.time_str = datestr(d_num, val);

        if isnumeric(val)
          tt.timeformat = set(tt.timeformat, 'format_nr', val);
        else
          tt.timeformat = set(tt.timeformat, 'format_str', val);
        end
      end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Set the timezone AND compute the current time string.
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    elseif (strcmp(prop, 'timezone')) && (compute)

      available_IDs = cell(java.util.TimeZone.getAvailableIDs);

      if strcmp (class(val), 'sun.util.calendar.ZoneInfo')
        tt.timezone = val;
      elseif ischar(val) && ismember(val, available_IDs)
        tt.timezone = java.util.TimeZone.getTimeZone(val);
      else
        error (sprintf(['### Please use a timezone ID\n'...
          'USE: get(time, ''timezone_IDs'') OR a java TimeZone']));
      end

      t_format = java.text.SimpleDateFormat(tt.timeformat.default_java_str);
      t_format.setTimeZone(tt.timezone);
      tt.time_str = char(t_format.format(tt.utc_epoch_milli));

      % Set the right time format
      if ~isempty(tt.timeformat.format_str)
        tt = set(tt, 'timeformat', tt.timeformat.format_str);
      elseif tt.timeformat.format_nr >= 0
        tt = set(tt, 'timeformat', tt.timeformat.format_nr);
      else
        error('### No time format is set. Should not happen.');
      end

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

```
    % Set the time_str AND compute the utc_epoch_milli
    %%%%
```

```
    % Set the unix epoche time AND compute the time string
    %%%%
```

```
            '\d{4}\.\d{2}\.\d{2}', 'yyyy.mm.dd'; ...
            '\d{2}-\d{2}',         'mm-dd';      ...
            '\.\d{1,3}',           '.FFF'};

  for ii = 1:size(parse,1)

    curr_parse = parse{ii,1};
    [match] = regexp(ret, curr_parse, 'match');

    if ~isempty(match) && length(match) == 1
      replace = parse{ii,2};
      ret = strrep(ret, match{1}, replace);
    end

  end

end

function default_plist = GET_DEFAULT_PLIST
  default_plist =  plist('name',            '', ...
                         'utc_epoch_milli', [], ...
                         'timezone',        java.util.TimeZone.getTimeZone
('UTC'), ...
                         'timeformat',      timeformat(), ...
                         'time_str',        '',           ...
                         'plist',           plist(),      ...
                         'created',         time(),       ...
                         'version',         '');
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: specwin/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: specwin/private   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help specwin/private/fromxml">specwin/private/fromxml</a>
-  read from part of an XML tree and create a specwin object.
   <a href="matlab:help specwin/private/get_window">specwin/private/get_window</
a>      -  returns the required window function as a structure.
   <a href="matlab:help specwin/private/kaiser_alpha">specwin/private/kaiser_alpha</
a>    -  returns the alpha parameter that gives the required input
   <a href="matlab:help specwin/private/kaiser_flatness">specwin/private/
kaiser_flatness</a> -  returns the flatness in dB of the central bin of a kaiser
   <a href="matlab:help specwin/private/kaiser_nenbw">specwin/private/kaiser_nenbw</
a>    -  returns the normalized noise-equivalent bandwidth for a
   <a href="matlab:help specwin/private/kaiser_rov">specwin/private/kaiser_rov</
a>      -  returns the recommended overlap for a Kaiser window with
   <a href="matlab:help specwin/private/kaiser_w3db">specwin/private/kaiser_w3db</
a>     -  returns the 3dB bandwidth in bins of a kaiser window with
   <a href="matlab:help specwin/private/retrieve">specwin/private/retrieve</
a>        -  specwin objects with given ids from the LTPDA repository specified by
the input
   <a href="matlab:help specwin/private/win_BH92">specwin/private/win_BH92</
a>        -  return BH92 window sample for input z.
   <a href="matlab:help specwin/private/win_Bartlett">specwin/private/win_Bartlett</
a>    -  return Bartlett window sample for input z.
   <a href="matlab:help specwin/private/win_FTHP">specwin/private/win_FTHP</
a>        -  return FTHP window sample for input z.
   <a href="matlab:help specwin/private/win_FTNI">specwin/private/win_FTNI</
a>        -  return FTNI window sample for input z.
   <a href="matlab:help specwin/private/win_FTSRS">specwin/private/win_FTSRS</
a>       -  return FTSRS window sample for input z.
   <a href="matlab:help specwin/private/win_HFT116D">specwin/private/win_HFT116D</
a>     -  return HFT116D window sample for input z.
   <a href="matlab:help specwin/private/win_HFT144D">specwin/private/win_HFT144D</
a>     -  return HFT144D window sample for input z.
   <a href="matlab:help specwin/private/win_HFT169D">specwin/private/win_HFT169D</
a>     -  return HFT169D window sample for input z.
```

```
   <a href="matlab:help specwin/private/win_HFT196D">specwin/private/win_HFT196D</
a>     -  return HFT196D window sample for input z.
   <a href="matlab:help specwin/private/win_HFT223D">specwin/private/win_HFT223D</
a>     -  return HFT223D window sample for input z.
   <a href="matlab:help specwin/private/win_HFT248D">specwin/private/win_HFT248D</
a>     -  return HFT248D window sample for input z.
   <a href="matlab:help specwin/private/win_HFT70">specwin/private/win_HFT70</
a>       -  return HFT70 window sample for input z.
   <a href="matlab:help specwin/private/win_HFT90D">specwin/private/win_HFT90D</
a>      -  return HFT90D window sample for input z.
   <a href="matlab:help specwin/private/win_HFT95">specwin/private/win_HFT95</
a>       -  return HFT95 window sample for input z.
   <a href="matlab:help specwin/private/win_Hamming">specwin/private/win_Hamming</
a>     -  return Hamming window sample for input z.
   <a href="matlab:help specwin/private/win_Hanning">specwin/private/win_Hanning</
a>     -  return Hanning window sample for input z.
   <a href="matlab:help specwin/private/win_Nuttall3">specwin/private/win_Nuttall3</
a>    -  return Nuttall3 window sample for input z.
   <a href="matlab:help specwin/private/win_Nuttall3a">specwin/private/
win_Nuttall3a</a>   -  return Nuttall3a window sample for input z.
   <a href="matlab:help specwin/private/win_Nuttall3b">specwin/private/
win_Nuttall3b</a>   -  return Nuttall3b window sample for input z.
   <a href="matlab:help specwin/private/win_Nuttall4">specwin/private/win_Nuttall4</
a>    -  return Nuttall4 window sample for input z.
   <a href="matlab:help specwin/private/win_Nuttall4a">specwin/private/
win_Nuttall4a</a>   -  return Nuttall4a window sample for input z.
   <a href="matlab:help specwin/private/win_Nuttall4b">specwin/private/
win_Nuttall4b</a>   -  return Nuttall4b window sample for input z.
   <a href="matlab:help specwin/private/win_Nuttall4c">specwin/private/
win_Nuttall4c</a>   -  return Nuttall4c window sample for input z.
   <a href="matlab:help specwin/private/win_Rectangular">specwin/private/
win_Rectangular</a> -  return rectangular window sample for input z.
   <a href="matlab:help specwin/private/win_SFT3F">specwin/private/win_SFT3F</
a>       -  return SFT3F window sample for input z.
   <a href="matlab:help specwin/private/win_SFT3M">specwin/private/win_SFT3M</
a>       -  return SFT3M window sample for input z.
   <a href="matlab:help specwin/private/win_SFT4F">specwin/private/win_SFT4F</
a>       -  return SFT4F window sample for input z.
   <a href="matlab:help specwin/private/win_SFT4M">specwin/private/win_SFT4M</
a>       -  return SFT4M window sample for input z.
   <a href="matlab:help specwin/private/win_SFT5F">specwin/private/win_SFT5F</
a>       -  return SFT5F window sample for input z.
   <a href="matlab:help specwin/private/win_SFT5M">specwin/private/win_SFT5M</
a>       -  return SFT5M window sample for input z.
   <a href="matlab:help specwin/private/win_Welch">specwin/private/win_Welch</
a>       -  return Welch window sample for input z.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get timespan properties.

SYNOPSIS ^

function val = get(ts, prop_name)

DESCRIPTION ^

```
 GET get timespan properties.

%%%
```

```
function val = get(ts, prop_name)
% GET get timespan properties.
%
%%%
```

ltpda_unwrapdeg

PURPOSE ^

LTPDA_UNWRAPDEG Unwrap a phase vector given in degrees.

SYNOPSIS ^

function deg = ltpda_unwrapdeg(phase)

DESCRIPTION ^

```
 LTPDA_UNWRAPDEG Unwrap a phase vector given in degrees.

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

 DESCRIPTION: LTPDA_UNWRAPDEG Unwrap a phase vector given in degrees.

 CALL:        deg = ltpda_unwrapdeg(phase)

 INPUTS:      phase - phase vector

 OUTPUTS:     deg   - degrees vector

 VERSION: $Id: ltpda_unwrapdeg.m,v 1.5 2007/07/13 12:17:39 ingo Exp $

 HISTORY: 26-01-2007 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- phase PHASE return the phase in degrees for a given complex input.

This function is called by:

SOURCE CODE ^

```
0001 function deg = ltpda_unwrapdeg(phase)
0002 % LTPDA_UNWRAPDEG Unwrap a phase vector given in degrees.
0003 %
```

```
%%%
```

set

PURPOSE ^

SET set timeformat properties.

SYNOPSIS ^

function tf = set(tf, varargin)

DESCRIPTION ^

```
 SET set timeformat properties.

%%%
```

SOURCE CODE ^

```
function tf = set(tf, varargin)
% SET set timeformat properties.
%
%%%
```

```
end

%%%
```

```
          tf.format_str = '';
        end
      else
        error('### Please use for %s a matlab format-number', tf.(field));
      end
    elseif (strcmp(prop, {'default_matlab_str', ...
        'default_matlab_nr',  ...
        'default_java_str'}))
      error('### Do not set one of the default values. Only the time constructor should set this values.');
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Set all other properties
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    else
      try
        tf.(prop) = val;
      catch
        error('### ''%s'' is not a valid %s-object property.', prop, class(tf));
      end
    end

  end

else
  display(tf);
  error('\n### No output variable!\n### Please use: %s = set(%s, ''key'', ''value'');', inputname(1), inputname(1));
end


function default_plist = GET_DEFAULT_PLIST
default_plist =  plist('format_str',         '', ...
                       'format_nr',          -1, ...
                       'default_matlab_str', '', ...
                       'default_matlab_nr',  -1, ...
                       'default_java_str',   '');
```

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/display">time/display</a>  -  overloads display
functionality for time objects.
   <a href="matlab:help time/eq">time/eq</a>       -  overloads the == operator for
time objects.
   <a href="matlab:help time/format">time/format</a>   -  Returns the time in the
handover format.
   <a href="matlab:help time/get">time/get</a>      -  get time properties.
   <a href="matlab:help time/isfield">time/isfield</a>  -  tests if the given field
is one of the object properties.
   <a href="matlab:help time/minus">time/minus</a>    -  overloads - operator for
time objects.
   <a href="matlab:help time/ne">time/ne</a>       -  overloads the ~= operator for
time objects.
   <a href="matlab:help time/plus">time/plus</a>     -  overloads - operator for
time objects.
   <a href="matlab:help time/save">time/save</a>     -  a time object to file.
   <a href="matlab:help time/set">time/set</a>      -  set time properties.
   <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/subsref">time/subsref</a>  -  Define field name
indexing for time objects.
   <a href="matlab:help time/time">time/time</a>     -  time object class
constructor.
   <a href="matlab:help time/xml">time/xml</a>      -  convert a time object to an
DOM XML object.
   <a href="matlab:help time/xmladd">time/xmladd</a>   -  add a time object to the
given node of the input XML DOM
   <a href="matlab:help time/xmlparse">time/xmlparse</a> -  creates a time from a
time xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get tsdata properties.

SYNOPSIS ^

function val = get(ts, propName)

DESCRIPTION ^

```
 GET get tsdata properties.

%%%
```

SOURCE CODE ^

```
function val = get(ts, propName)
% GET get tsdata properties.
%
%%%
```

ltpda_user_classes

PURPOSE ^

ltpda_user_classes lists all the LTPDA user object types.

SYNOPSIS ^

function classes = ltpda_user_classes(varargin)

DESCRIPTION ^

```
 ltpda_user_classes lists all the LTPDA user object types.

%%%
```

```
% ltpda_user_classes lists all the LTPDA user object types.
%
%%%
```

set

PURPOSE ^

SET set timespan properties.

SYNOPSIS ^

function ts = set(ts, varargin)

DESCRIPTION ^

```
 SET set timespan properties.

%%%
```

- set SET set timespan properties.
- timespan TIMESPAN timespan object class constructor.

SOURCE CODE ^

```
function ts = set(ts, varargin)
% SET set timespan properties.
%
%%%
```

```
end

%%% Check if this is a special call: default parameter list, cvs-version, category
if ~isempty(varargin{1}) == 1 && ischar(varargin{1})
  in = varargin{1};
  if strcmp(in, 'Params')
    ts = DEFAULT_PLIST;
    return
  elseif strcmp(in, 'Version')
    ts = VERSION;
    return
  elseif strcmp(in, 'Category')
    ts = CATEGORY;
    return
  end
end

%%%
```

```
          ts_format = char(ts.timeformat);
          if isnumeric(str2double(ts_format))
            ts_format = str2double(ts_format);
          end
          tt = time(val, ts_format);
          tt = set(tt, 'timeformat', ts.timeformat);
          ts.(prop) = tt;
        end

        % Compute the interval
        ts = compute_interval(ts);

      %%%%
```

```
error('\n### No output variable!\n### Please use: %s = set(%s, ''key'', ''value'');', inputname(1), inputname(1));
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: time/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%   class: time/private   %%%%%%%%%%%%%%%%%%

   <a href="matlab:help time/private/fromxml">time/private/fromxml</a>  -  read from
part of an XML tree and create a time object.
   <a href="matlab:help time/private/retrieve">time/private/retrieve</a> -  time
objects with given ids from the LTPDA repository specified by the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET gets xydata properties.

SYNOPSIS ^

function val = get(xy, propName)

DESCRIPTION ^

```
 GET gets xydata properties.

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

 DESCRIPTION: GET gets xydata properties.

 CALL:    name    = get(xy, 'name');
          x       = get(xy, 'x');
          y       = get(xy, 'y');
          xunits  = get(xy, 'xunits');
          yunits  = get(xy, 'yunits');
          created = get(xy, 'created');
          version = get(xy, 'version');

 VERSION: $Id: get.m,v 1.5 2008/02/11 17:28:46 ingo Exp $

 HISTORY: 30-01-2007 Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- char CHAR convert a param object into a string.
- display DISPLAY overloads display functionality for xydata objects.

SOURCE CODE ^

```
function val = get(xy, propName)
% GET gets xydata properties.
%
%%%
```

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

%%%
```

```
 The following call returns a string that contains the routine category:

 >> category = ltpda_foo('Category')

%%%
```

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

```
  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}),...
    'mfilename', ALGONAME, ...
    'description', find(pl,'description'));

  bs = [bs b];

end


%% Reshape the ouput to the same size of the input
varargout{1} = reshape(bs, size(as));


```

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

set

PURPOSE ^

SET sets a tsdata property.

SYNOPSIS ^

function ts = set(ts, varargin)

DESCRIPTION ^

```
 SET sets a tsdata property.

%%%
```

```
function ts = set(ts, varargin)
% SET sets a tsdata property.
%
%%%
```

```

if (nargout == 0)
  display(ts);
  error('\n### No output variable!\n### Please use: %s = set(%s, ''key'', ''value'');', inputname(1), inputname(1));
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: timeformat %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%   class: timeformat   %%%%%%%%%%%%%%%%%%

   <a href="matlab:help timeformat/char">timeformat/char</a>       -  convert a
timeformat object into a string.
   <a href="matlab:help timeformat/display">timeformat/display</a>    -  overloads
display functionality for timeformat objects.
   <a href="matlab:help timeformat/eq">timeformat/eq</a>         -  overloads the ==
operator for timeformat objects.
   <a href="matlab:help timeformat/isfield">timeformat/isfield</a>    -  tests if
the given field is one of the object properties.
   <a href="matlab:help timeformat/ne">timeformat/ne</a>         -  overloads the ~=
operator for timeformat objects.
   <a href="matlab:help timeformat/save">timeformat/save</a>       -  a timeformat
object to file.
   <a href="matlab:help timeformat/set">timeformat/set</a>        -  set timeformat
properties.
   <a href="matlab:help timeformat/subsref">timeformat/subsref</a>    -  Define
field name indexing for timeformat objects.
   <a href="matlab:help timeformat/timeformat">timeformat/timeformat</a> -
timeformat object class constructor.
   <a href="matlab:help timeformat/xml">timeformat/xml</a>        -  convert a
timeformat object to an DOM XML object.
   <a href="matlab:help timeformat/xmladd">timeformat/xmladd</a>     -  add a time
object to the given node of the input XML DOM
   <a href="matlab:help timeformat/xmlparse">timeformat/xmlparse</a>   -  creates a
timeformat from a timeformat xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET gets xyzdata properties.

SYNOPSIS ^

function val = get(xyz, propName)

DESCRIPTION ^

```
 GET gets xyzdata properties.

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

 DESCRIPTION: GET gets xyzdata properties.

 CALL:    name    = get(xyz, 'name');
          x       = get(xyz, 'x');
          y       = get(xyz, 'y');
          z       = get(xyz, 'z');
          xunits  = get(xyz, 'xunits');
          yunits  = get(xyz, 'yunits');
          zunits  = get(xyz, 'zunits');
          created = get(xyz, 'created');
          version = get(xyz, 'version');

 VERSION: $Id: get.m,v 1.2 2008/02/11 17:28:46 ingo Exp $

 HISTORY: 30-01-2007 Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- display DISPLAY overloads display functionality for xyzdata objects.

SOURCE CODE ^

```
function val = get(xyz, propName)
% GET gets xyzdata properties.
%
%%%
```

ltpda_utc2gps

PURPOSE ^

LTPDA_UTC2GPS Converts UTC time to GPS seconds.

SYNOPSIS ^

function GPS_time=ltpda_utc2gps(UTC_time)

DESCRIPTION ^

```
 LTPDA_UTC2GPS Converts UTC time to GPS seconds.

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

 DESCRIPTION: LTPDA_UTC2GPS Converts UTC time to GPS seconds.
              UTC_time can also be an array of UTC times.

 CALL:        GPS_time=ltpda_UTC2GPS(UTC_time)

 FORMAT:      UTC time format: 'yyy-mm-dd- HH:MM:SS'
              GPS time format: Seconds since 6. January 1980

    EXAMPLES:    GPS_time=UTC2GPS('2002-07-19 16:00:00')
                 GPS_time=711129613

                 GPS_time=UTC2GPS(['2002-07-19 16:00:00';'2001-07-19 16:00:00'])
                 GPS_time=[711129613 ; 679593613]

 VERSION:     $Id: ltpda_utc2gps.m,v 1.3 2007/07/16 12:52:21 ingo Exp $

 HISTORY:     xx-xx-xxxx Karsten Koetter
                 Creation
              24-01-2007 M Hewitson.
                 Maintained

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function GPS_time=ltpda_utc2gps(UTC_time)
% LTPDA_UTC2GPS Converts UTC time to GPS seconds.
%
%%%
```

set

PURPOSE ^

SET sets a xydata property.

SYNOPSIS ^

function xy = set(xy, varargin)

DESCRIPTION ^

```
 SET sets a xydata property.

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

 DESCRIPTION: SET sets a xydata property.

 CALL: c = set(c, 'name',    'my_name');
       c = set(c, 'x',        X);
       c = set(c, 'x',       [1 2 3]);
       c = set(c, 'y',        Y);
       c = set(c, 'y',       [1 2 3]);
       c = set(c, 'xunits',  'sec');
       c = set(c, 'yunits',  'V');
       c = set(c, 'created',  time());
       c = set(c, 'version', 'xydata/set.m,v 1.7...');

 VERSION: $Id: set.m,v 1.8 2008/02/12 15:20:33 ingo Exp $

 HISTORY: 30-01-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- display DISPLAY overloads display functionality for xydata objects.

This function is called by:

SOURCE CODE ^

```
function xy = set(xy, varargin)
% SET sets a xydata property.
%
%%%
```

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: timeformat/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%   class: timeformat/private   %%%%%%%%%%%%%%%%%%

   <a href="matlab:help timeformat/private/fromxml">timeformat/private/fromxml</a>
-  read from part of an XML tree and create a timeformat object.
   <a href="matlab:help timeformat/private/retrieve">timeformat/private/retrieve</a>
-  timeformat objects with given ids from the LTPDA repository specified by the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get

PURPOSE ^

GET get zero properties.

SYNOPSIS ^

function val = get(p, propName)

DESCRIPTION ^

```
 GET get zero properties.

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

 DESCRIPTION: GET get zero properties.

 CALL:        name    = get(z, 'name');
              f       = get(z, 'f');
              q       = get(z, 'q');
              ri      = get(z, 'ri');
              version = get(z, 'version');
              created = get(z, 'created');

 VERSION:     $Id: get.m,v 1.5 2008/02/11 17:28:46 ingo Exp $

 HISTORY:     30-01-2007 Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- cz2iir CZ2IIR return a,b IIR filter coefficients for a complex zero designed using the bilinear transform.
- display DISPLAY overloads display functionality for zero objects.
- resp RESP returns the complex response of the zero object.
- rz2iir RZ2IIR Return a,b IIR filter coefficients for a real zero designed using the bilinear transform.

SOURCE CODE ^

```
function val = get(p, propName)
% GET get zero properties.
%
%%%
```

ltpda_versions

PURPOSE ^

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
versions = {...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/abs.m', 'abs', '1.13', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/', 'fd6fffc433a849b827ad588146c53ea2'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/acos.m', 'acos', '1.11', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/', 'a6956a091ccdad77e1910cabdd62bb98'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/ao.m', 'ao', '1.77', '2008/02/18', '13:34:48', 'ingo', 'classes/@', '3ac7a34295fd99a8a50d7b279c598520'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/ao2m.m', 'ao2m', '1.13', '2008/02/12', '19:29:33', 'hewitson', 'classes/@ao/', '79fd1a4fac9fec79afab84dfdfc76c82'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/asin.m', 'asin', '1.10', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/', 'dfa91e419d504d0329949af552467e2a'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/atan.m', 'atan', '1.10', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/', '23feeb76d4320eff7f61f0193c919c9b'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/attachm.m', 'attachm', '1.5', '2008/01/22', '20:59:42', 'hewitson', 'classes/@ao/', '8e8677a6b79082950368829905cff917'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/attachmdl.m', 'attachmdl', '1.4', '2008/01/22', '20:59:42', 'hewitson', 'classes/@ao/', 'caa8940bad9b7ec6f1faf3ad3d2b31f9'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/cat.
```

```
m', 'cat', '1.7', '2008/02/12', '10:11:13', 'mauro', 'classes/@ao/',
'14775d3e0864c830663913f7b447da40'; ...
0011        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/char.
m', 'char', '1.10', '2008/02/12', '19:29:33', 'hewitson', 'classes/@ao/',
'efb9e9be2f6f7b48f5eb3d14f827ef20'; ...
0012        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
complex.m', 'complex', '1.8', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/',
'71a2ab107c28936557516026a87df02f'; ...
0013        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/conj.
m', 'conj', '1.14', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/',
'a51ab2b494da3d403d5ab1e870a5ecdb'; ...
0014        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/cos.
m', 'cos', '1.10', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/',
'0e43b8ed316da3202299f9a96fee4704'; ...
0015        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
ctranspose.m', 'ctranspose', '1.15', '2008/02/12', '08:31:45', 'mauro', 'classes/
@ao/', '635f6882a15183ee25bce170717c1a57'; ...
0016        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/demux.
m', 'demux', '1.7', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/',
'9763a2387230680f10181121493c8980'; ...
0017        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/det.
m', 'det', '1.17', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/',
'd85a7b4fe7c1446cc4a48e5b0d0eaa88'; ...
0018        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/diag.
m', 'diag', '1.19', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/',
'2d4a029ad809fe269ab60f0a66d9d0c5'; ...
0019        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
display.m', 'display', '1.17', '2008/02/12', '19:29:33', 'hewitson', 'classes/@ao/',
'398199cf773df9b9153bc503b1ce27cb'; ...
0020        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
downsample.m', 'downsample', '1.2', '2008/02/12', '08:31:45', 'mauro', 'classes/
@ao/', '40ddd3773a8cba2611e52d328d4c53d2'; ...
0021        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/eig.
m', 'eig', '1.20', '2008/02/19', '15:23:23', 'mauro', 'classes/@ao/',
'06871a29756247346acf0f29222156af'; ...
0022        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/eq.m',
'eq', '1.12', '2008/02/18', '14:41:13', 'ingo', 'classes/@ao/',
'26cdbe64dc148dbe03bddec87e33c39f'; ...
0023        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/exp.
m', 'exp', '1.14', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/',
'e6730d4af6d4eebcf7fcbaaac7ccac00'; ...
0024        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/export.
m', 'export', '1.7', '2008/01/22', '20:59:42', 'hewitson', 'classes/@ao/',
'a242dc9a021354d6ec655a7d636fa312'; ...
0025        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/fft.
m', 'fft', '1.12', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/',
'0f3b7549e4afc0240fb2c7ca89c407f7'; ...
0026        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/filter.
m', 'filter', '1.26', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/',
'64b2356729fd5f31996b4649bf71d674'; ...
0027        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
filtfilt.m', 'filtfilt', '1.15', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/',
'06ece4b7fb0546c17f5cc9f57c573d59'; ...
```

```
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/find.m', 'find', '1.13', '2008/02/19', '18:01:59', 'ingo', 'classes/@ao/', 'f0bb19d7ddcfb7544ce3e67bb05e3a88'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/ge.m', 'ge', '1.6', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/', '08cc0e0b4ebea9359017967a83b5765e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/get.m', 'get', '1.15', '2008/02/13', '07:58:13', 'mauro', 'classes/@ao/', 'd24270c6583a9de62e3bbba7206f6ab2'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/gt.m', 'gt', '1.5', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/', 'f45844547edd60f90bbf8c054aa37613'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/hist.m', 'hist', '1.10', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/', '317a1ca9e6d0a810fe2b24b30dd2e92c'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/imag.m', 'imag', '1.5', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/', '94c96a7eb5239627c44f4d74025e829a'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/index.m', 'index', '1.8', '2008/02/12', '19:29:34', 'hewitson', 'classes/@ao/', 'ac096319e91c0da1a234dee7bd9aec19'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/interp.m', 'interp', '1.11', '2008/02/12', '08:31:45', 'mauro', 'classes/@ao/', '842a52a7cfa3765739e8571392d4b6bf'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/inv.m', 'inv', '1.16', '2008/02/12', '19:29:33', 'hewitson', 'classes/@ao/', '67df837b08969791eae30ca33f421335'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/iplot.m', 'iplot', '1.25', '2008/02/12', '09:29:24', 'mauro', 'classes/@ao/', '955dc163a69fdab3e7904a327d1e9442'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/isfield.m', 'isfield', '1.9', '2008/02/12', '09:29:24', 'mauro', 'classes/@ao/', '93b871725d192bbfef1bbd2ab5bd5f91'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/isvalid.m', 'isvalid', '1.5', '2008/02/14', '13:48:33', 'ingo', 'classes/@ao/', 'e65e011b424cb7f0c1c97ab34ec521c2'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/join.m', 'join', '1.13', '2008/02/13', '18:35:22', 'hewitson', 'classes/@ao/', '05e63c1174b9983d5e2581beec04bfbc'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/le.m', 'le', '1.7', '2008/02/12', '14:31:27', 'hewitson', 'classes/@ao/', '3655801ee87ed78ffa38758a024359d1'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/len.m', 'len', '1.10', '2008/02/12', '19:29:33', 'hewitson', 'classes/@ao/', '06ec3cb2fd86b2f13379eaf84b256040'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/ln.m', 'ln', '1.7', '2008/02/12', '09:29:24', 'mauro', 'classes/@ao/', '3fb82858bd5ce9658110dfafb3d465fb'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/log.m', 'log', '1.13', '2008/02/12', '09:29:24', 'mauro', 'classes/@ao/', '302ba63597155c4973bd4165895e7a67'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/log10.m', 'log10', '1.10', '2008/02/12', '09:29:24', 'mauro', 'classes/@ao/',
```

```
'7a59e721d1ddd9d1ee9da8fba68df983'; ...
0046         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/lt.m',
'lt', '1.7', '2008/02/12', '14:31:27', 'hewitson', 'classes/@ao/',
'762df72bddee3940c744c91f4ee2eeb5'; ...
0047         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/md5.
m', 'md5', '1.5', '2008/02/19', '19:45:58', 'hewitson', 'classes/@ao/',
'137fadd26448720bd3d87746b13d04d6'; ...
0048         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/mean.
m', 'mean', '1.14', '2008/02/19', '17:26:40', 'mauro', 'classes/@ao/',
'd0734752ac6889bba8ff5bba310efb58'; ...
0049         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/median.
m', 'median', '1.11', '2008/02/19', '17:26:40', 'mauro', 'classes/@ao/',
'4da53bed9a53cdd14a2a5dd77ab60576'; ...
0050         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/minus.
m', 'minus', '1.12', '2008/02/13', '18:35:22', 'hewitson', 'classes/@ao/',
'23c0583d67e6d8812f7be53a4c3b071f'; ...
0051         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/mpower.
m', 'mpower', '1.13', '2008/02/13', '18:35:22', 'hewitson', 'classes/@ao/',
'57733563a8ebd6feb4b503ba237faff4'; ...
0052         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
mrdivide.m', 'mrdivide', '1.13', '2008/02/13', '18:35:22', 'hewitson', 'classes/
@ao/', '8aaae8cfd702aebde30796d45bdd77ea'; ...
0053         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/mtimes.
m', 'mtimes', '1.14', '2008/02/13', '18:35:22', 'hewitson', 'classes/@ao/',
'661a1a3f433bc5befc7126480a2151d2'; ...
0054         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/mux.
m', 'mux', '1.6', '2008/02/12', '09:29:24', 'mauro', 'classes/@ao/',
'290d5392db81d8a72e8dcf7d1e17dd5c'; ...
0055         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/ne.m',
'ne', '1.8', '2008/02/18', '15:12:58', 'ingo', 'classes/@ao/',
'8d48593c944c19db5e2a1f25dd25ac1b'; ...
0056         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/norm.
m', 'norm', '1.16', '2008/02/12', '09:29:24', 'mauro', 'classes/@ao/',
'6cfd55af9efa5e4566b072b21a94130d'; ...
0057         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/phase.
m', 'phase', '1.6', '2008/02/12', '09:29:24', 'mauro', 'classes/@ao/',
'5622d3a60070ae72bdffa43d351b6329'; ...
0058         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/plot.
m', 'plot', '1.31', '2008/02/12', '09:43:32', 'mauro', 'classes/@ao/',
'3cd6a60a79043ade0a722108bd1e9c69'; ...
0059         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/plus.
m', 'plus', '1.16', '2008/02/13', '18:35:22', 'hewitson', 'classes/@ao/',
'4baba31295408494aa94425757ea0bff'; ...
0060         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
polyfit.m', 'polyfit', '1.11', '2008/02/12', '09:43:32', 'mauro', 'classes/@ao/',
'6840a825275d8839d599581f35c66010'; ...
0061         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/power.
m', 'power', '1.11', '2008/02/15', '16:27:49', 'hewitson', 'classes/@ao/',
'75e6866c37674ad74ca5a413f1a7165e'; ...
0062         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
private/aooperate.m', 'aooperate', '1.18', '2008/02/13', '18:05:44', 'hewitson',
'classes/@ao/private/', 'bf38369b50e811e7dfc8432a5068848d'; ...
0063         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
```

```
private/collect_inputs.m', 'collect_inputs', '1.1', '2007/11/28', '11:38:46',
'ingo', 'classes/@ao/private/', 'c86ddb1d201f779eeac9d2083f970548'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
private/retrieve.m', 'retrieve', '1.9', '2007/08/31', '17:40:08', 'hewitson',
'classes/@ao/private/', '280752ac5c22bd0bb8c2ed264f4ad5bc'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
private/xdoc2ao.m', 'xdoc2ao', '1.2', '2007/11/14', '16:30:18', 'ingo', 'classes/@ao/
private/', '23c0c88566ee9f22fa4be9a000f38578'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
rdivide.m', 'rdivide', '1.13', '2008/02/13', '18:35:22', 'hewitson', 'classes/@ao/',
'125bb7e61d4d81c24908054dd59215f3'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/real.
m', 'real', '1.5', '2008/02/12', '09:43:32', 'mauro', 'classes/@ao/',
'd52fe761e30397fc77e5d24fd3582734'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
resample.m', 'resample', '1.14', '2008/02/12', '09:43:32', 'mauro', 'classes/@ao/',
'8da922866d7c343885b96280a3a21672'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/save.
m', 'save', '1.16', '2008/02/13', '10:13:53', 'ingo', 'classes/@ao/',
'b1b8c5a886fc02df0cc72c953131d17b'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/select.
m', 'select', '1.16', '2008/02/12', '09:43:32', 'mauro', 'classes/@ao/',
'6f266d915730f2ab03b62f4c09d55a6d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/set.
m', 'set', '1.31', '2008/02/19', '16:29:38', 'ingo', 'classes/@ao/',
'a6205bf39b9f8420d5377ea8442d80d0'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/setnh.
m', 'setnh', '1.8', '2008/02/19', '16:29:38', 'ingo', 'classes/@ao/',
'09f6d151e7fc3cdc13b24a9a26c11f5c'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/sin.
m', 'sin', '1.18', '2008/02/12', '10:11:13', 'mauro', 'classes/@ao/',
'c1f7ba588117207a863c6d4018d9a158'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
spectrogram.m', 'spectrogram', '1.2', '2008/02/12', '10:11:13', 'mauro', 'classes/
@ao/', '377e51e0899a1021cd726857a302d9ed'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/split.
m', 'split', '1.32', '2008/02/12', '10:11:13', 'mauro', 'classes/@ao/',
'2bb28852010cbeecb5c18e825f119bdf'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/sqrt.
m', 'sqrt', '1.15', '2008/02/12', '10:11:13', 'mauro', 'classes/@ao/',
'5223b3d2649b2203296682e910e3bfd7'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/stairs.
m', 'stairs', '1.8', '2008/02/12', '10:11:13', 'mauro', 'classes/@ao/',
'c1944b3111431c33e635b39c78495fad'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/std.
m', 'std', '1.11', '2008/02/19', '14:44:38', 'ingo', 'classes/@ao/',
'c539286d6b0eedf67fa4fe9fed1c440c'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/string.
m', 'string', '1.5', '2008/02/12', '19:29:33', 'hewitson', 'classes/@ao/',
'43914a960d294db7a9eebcc1f5f6a853'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
subsasgn.m', 'subsasgn', '1.6', '2008/02/12', '19:29:33', 'hewitson', 'classes/
@ao/', '262a3ce57b8ce9f3cb3593599a244427'; ...
```

```
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
subsref.m', 'subsref', '1.16', '2008/02/12', '19:29:33', 'hewitson', 'classes/@ao/',
'26728ae5e946afea1740d51beee6b3af'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/sum.
m', 'sum', '1.9', '2008/02/19', '14:15:34', 'ingo', 'classes/@ao/',
'd5db954dd18b4609eec3fafacb98ee70'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/svd.
m', 'svd', '1.15', '2008/02/12', '10:11:13', 'mauro', 'classes/@ao/',
'53141f133b96eb521489f5e80bf604ad'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/tan.
m', 'tan', '1.11', '2008/02/12', '10:11:13', 'mauro', 'classes/@ao/',
'16b14adbafe854786326aced6bea4006'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/times.
m', 'times', '1.13', '2008/02/13', '18:35:22', 'hewitson', 'classes/@ao/',
'6d9467a567f02bb8879b4cdf76ff4fd6'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
transpose.m', 'transpose', '1.14', '2008/02/12', '10:11:13', 'mauro', 'classes/
@ao/', '02bd1a315ba7946d0233c035199ebfbd'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/
upsample.m', 'upsample', '1.3', '2008/02/12', '10:11:13', 'mauro', 'classes/@ao/',
'3de69173b22bc42c0c6b0bd617e6f92e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@ao/var.
m', 'var', '1.11', '2008/02/19', '13:31:33', 'ingo', 'classes/@ao/',
'4e8516f79e7ca2ed33b99c44ce1eda28'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/
cdata.m', 'cdata', '1.21', '2008/02/19', '17:23:11', 'mauro', 'classes/@',
'41664c2e7ca2b4be41fce49b7e1b273c'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/
char.m', 'char', '1.4', '2008/02/12', '19:54:05', 'hewitson', 'classes/@cdata/',
'c88bef791df409640882413933aeacdd'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/
display.m', 'display', '1.7', '2008/02/12', '19:54:05', 'hewitson', 'classes/
@cdata/', 'babc14139196fc8213b1401ce266009a'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/eq.
m', 'eq', '1.15', '2008/02/18', '14:41:13', 'ingo', 'classes/@cdata/',
'5017ea7a879929f1d4375713fa4f0fd5'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/get.
m', 'get', '1.7', '2008/02/11', '17:28:45', 'ingo', 'classes/@cdata/',
'bb045609c28b525b974cf7b23db4e25d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/
get_xy_values.m', 'get_xy_values', '1.5', '2008/02/12', '19:54:05', 'hewitson',
'classes/@cdata/', 'e6b4edbb6e4f73742f47cb28011a7a43'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/
isfield.m', 'isfield', '1.4', '2008/02/15', '16:27:15', 'mauro', 'classes/@cdata/',
'9bd3c1dbe38d0a16e2cd1f93c96dd4c5'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/
isvalid.m', 'isvalid', '1.1', '2008/02/14', '13:48:33', 'ingo', 'classes/@cdata/',
'a2477020ec3e49be73fd183164e83698'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/ne.
m', 'ne', '1.4', '2008/02/18', '15:12:58', 'ingo', 'classes/@cdata/',
'c8daeeb6ff82d907b626100738438d71'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/
private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:08', 'hewitson',
```

```
'classes/@cdata/private/', 'e32da801b57ad46b825f441db96f35a2'; ...
0099         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/
save.m', 'save', '1.4', '2008/02/15', '16:27:15', 'mauro', 'classes/@cdata/',
'8d525950856673a4ebbaa12aff83598b'; ...
0100         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/set.
m', 'set', '1.11', '2008/02/12', '15:20:32', 'ingo', 'classes/@cdata/',
'15c7a8da645697ca382c09c537c832ac'; ...
0101         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/
set_xy_axis.m', 'set_xy_axis', '1.4', '2008/02/12', '19:54:05', 'hewitson', 'classes/
@cdata/', '01ceb7e1046c8be063798fe3759dd36b'; ...
0102         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/
single_operation.m', 'single_operation', '1.14', '2008/02/12', '19:54:05',
'hewitson', 'classes/@cdata/', '237530a27fa9ee8b3c6c822909a605e3'; ...
0103         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/
subsasgn.m', 'subsasgn', '1.5', '2008/02/13', '10:13:53', 'ingo', 'classes/@cdata/',
'0559306e3aec54fe0dfe6dfa75fcea31'; ...
0104         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@cdata/
subsref.m', 'subsref', '1.8', '2008/02/12', '19:54:05', 'hewitson', 'classes/
@cdata/', '423562471a8b0cfc5d29bd4e90d69c83'; ...
0105         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
char.m', 'char', '1.4', '2008/02/12', '22:11:21', 'hewitson', 'classes/@fsdata/',
'2adec1e52910016d746beffb358f099f'; ...
0106         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
display.m', 'display', '1.8', '2008/02/12', '22:11:21', 'hewitson', 'classes/
@fsdata/', '37b132bd50ca29e7646b2f8acf9ccab8'; ...
0107         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/eq.
m', 'eq', '1.9', '2008/02/18', '14:41:13', 'ingo', 'classes/@fsdata/',
'655b25148904a211abbd718ade8f29e0'; ...
0108         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
fsdata.m', 'fsdata', '1.20', '2008/02/13', '10:13:53', 'ingo', 'classes/@',
'4f8ac50004c0d927fae1cc1387b44f77'; ...
0109         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
get.m', 'get', '1.11', '2008/02/11', '17:28:45', 'ingo', 'classes/@fsdata/',
'afe24c03617a1e1dcfd266e77bc9bdc1'; ...
0110         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
get_xy_values.m', 'get_xy_values', '1.5', '2008/02/12', '22:11:21', 'hewitson',
'classes/@fsdata/', 'db86aa9181ac2f0e791aca9e163c268f'; ...
0111         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
isfield.m', 'isfield', '1.4', '2008/02/13', '13:47:53', 'mauro', 'classes/@fsdata/',
'38f3d5ee23b84c20865caafe95bea0b0'; ...
0112         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
isvalid.m', 'isvalid', '1.1', '2008/02/14', '13:48:33', 'ingo', 'classes/@fsdata/',
'209b5ad530e18de0ca6e90eb62fb04ba'; ...
0113         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/ne.
m', 'ne', '1.4', '2008/02/18', '15:12:58', 'ingo', 'classes/@fsdata/',
'5df693fc4dc0e01ddaae6a8365846b9b'; ...
0114         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:08', 'hewitson',
'classes/@fsdata/private/', '29dd0dac87b389f45db7de778ab2e2fc'; ...
0115         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
reshapeF.m', 'reshapeF', '1.5', '2008/02/12', '22:11:21', 'hewitson', 'classes/
@fsdata/', '93ab1f5e07cce1f77d9d71f4714397da'; ...
0116         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
```

```
save.m', 'save', '1.4', '2008/02/15', '16:27:15', 'mauro', 'classes/@fsdata/',
'f57ca855abaee0f475e655329f309523'; ...
0117         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
set.m', 'set', '1.14', '2008/02/12', '15:20:32', 'ingo', 'classes/@fsdata/',
'6c4dfb7c5b0d982d93a9d63680997968'; ...
0118         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
setFreq.m', 'setFreq', '1.5', '2008/02/12', '22:11:21', 'hewitson', 'classes/
@fsdata/', 'db8fe35c02d1d34eb02474f77cd63839'; ...
0119         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
set_xy_axis.m', 'set_xy_axis', '1.5', '2008/02/12', '22:11:21', 'hewitson', 'classes/
@fsdata/', '58050cc20fb367a89a5120610f61924e'; ...
0120         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
single_operation.m', 'single_operation', '1.12', '2008/02/12', '22:11:21',
'hewitson', 'classes/@fsdata/', '7cce9422800fdd40e9b6d7f64696c7a5'; ...
0121         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
subsasgn.m', 'subsasgn', '1.4', '2008/02/12', '22:11:21', 'hewitson', 'classes/
@fsdata/', '151969d770febd263065643e9ea1c900'; ...
0122         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@fsdata/
subsref.m', 'subsref', '1.14', '2008/02/12', '22:11:21', 'hewitson', 'classes/
@fsdata/', '2c09cb7a707d2ba7093ad3c7a58e7eb7'; ...
0123         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
char.m', 'char', '1.4', '2008/02/12', '22:30:27', 'hewitson', 'classes/@history/',
'2145f1a9654403163c3c3e07f8d87405'; ...
0124         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
display.m', 'display', '1.10', '2008/02/12', '22:30:27', 'hewitson', 'classes/
@history/', 'bb30d086d294067dc96df594e5aaa48d'; ...
0125         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
eq.m', 'eq', '1.6', '2008/02/18', '14:41:13', 'ingo', 'classes/@history/',
'fc7a24d1a71b80ccc2e7a8ec7660d6b8'; ...
0126         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
get.m', 'get', '1.7', '2008/02/11', '17:28:45', 'ingo', 'classes/@history/',
'1cf1321a6f66a2a9deefac4795dc78a4'; ...
0127         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
getNodes.m', 'getNodes', '1.11', '2008/02/12', '22:30:27', 'hewitson', 'classes/
@history/', 'b36f88359d666ac163ee8ea500438172'; ...
0128         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
getNodes_plot.m', 'getNodes_plot', '1.4', '2008/02/12', '22:30:27', 'hewitson',
'classes/@history/', '73eca3a77b9e20490f27495e8cfc2e2a'; ...
0129         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
hist2dot.m', 'hist2dot', '1.4', '2008/02/12', '22:30:27', 'hewitson', 'classes/
@history/', 'cef99a4c56b237bdb7d92cefd17d92ec'; ...
0130         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
hist2m.m', 'hist2m', '1.15', '2008/02/12', '22:30:27', 'hewitson', 'classes/
@history/', '2469633412fcb8843729d7d5222d95e0'; ...
0131         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
history.m', 'history', '1.19', '2008/02/13', '10:13:53', 'ingo', 'classes/@',
'760178b4348f96ca7a9a9d7ff4763a77'; ...
0132         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
isfield.m', 'isfield', '1.4', '2008/02/15', '16:27:15', 'mauro', 'classes/
@history/', 'f5246985fa867d372850c94279b3683e'; ...
0133         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
isvalid.m', 'isvalid', '1.1', '2008/02/14', '13:48:33', 'ingo', 'classes/@history/',
'9d2f1bf4fbc317c81ccff37c5e540a55'; ...
```

```
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
ne.m', 'ne', '1.4', '2008/02/18', '15:12:58', 'ingo', 'classes/@history/',
'7ebf238db24d8b530a807d13cae4d8b8'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
plot.m', 'plot', '1.17', '2008/02/12', '22:30:27', 'hewitson', 'classes/@history/',
'334bcb4ea6b3ebbd3e49c23d9eb55dce'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:08', 'hewitson',
'classes/@history/private/', '04281daf707597d75e2d43c9314728c2'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
save.m', 'save', '1.4', '2008/02/15', '16:27:15', 'mauro', 'classes/@history/',
'8afef772b42ca57091474cc9678692ab'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
set.m', 'set', '1.11', '2008/02/12', '15:20:32', 'ingo', 'classes/@history/',
'bf6fc16fc6910e9df7b9778415e7d948'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
string.m', 'string', '1.4', '2008/02/12', '22:30:27', 'hewitson', 'classes/
@history/', 'ff7254dfb34f34c3529f726a6112843e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
subsasgn.m', 'subsasgn', '1.3', '2008/02/12', '22:30:27', 'hewitson', 'classes/
@history/', 'e75bc2cc9144ffa8ee114160372e3f38'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@history/
subsref.m', 'subsref', '1.4', '2008/02/12', '22:30:27', 'hewitson', 'classes/
@history/', 'a7291d4cdaaddeb3ad85faf5bf06aa07'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/char.
m', 'char', '1.4', '2008/02/12', '22:39:49', 'hewitson', 'classes/@mfir/',
'b0a04b236b90f656e4813bc30c195fa8'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/
display.m', 'display', '1.8', '2008/02/12', '22:39:49', 'hewitson', 'classes/
@mfir/', '4d11c5f1912e02f1d6dfc59245a9e511'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/eq.
m', 'eq', '1.5', '2008/02/18', '14:41:13', 'ingo', 'classes/@mfir/',
'4f9ec33d2c7708a9c85ddfc24ce9d704'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/get.
m', 'get', '1.4', '2008/02/11', '17:28:45', 'ingo', 'classes/@mfir/',
'306a52878e739e3ba7e44a1934ec0748'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/
isfield.m', 'isfield', '1.4', '2008/02/15', '16:27:16', 'mauro', 'classes/@mfir/',
'61609a68a86a0369aece6e9965774b29'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/
isvalid.m', 'isvalid', '1.2', '2008/02/15', '17:36:45', 'ingo', 'classes/@mfir/',
'2f04ef6dccf23054d43270135089cc04'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/mfir.
m', 'mfir', '1.24', '2008/02/18', '06:49:17', 'mauro', 'classes/@',
'16387dddace40083f2e918e8af36a778'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/ne.
m', 'ne', '1.4', '2008/02/18', '15:12:58', 'ingo', 'classes/@mfir/',
'd57692b8c10e9b0848e33e5875ec3fea'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/
private/mkbandpass.m', 'mkbandpass', '1.4', '2007/10/24', '10:55:11', 'ingo',
'classes/@mfir/private/', '407284aaa095e13530db397ba9d92cd3'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/
private/mkbandreject.m', 'mkbandreject', '1.4', '2007/10/24', '10:55:11', 'ingo',
```

```
'classes/@mfir/private/', 'd1b3268884d22eeecf8045d70c5c9388'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/
private/mkhighpass.m', 'mkhighpass', '1.4', '2007/10/24', '10:55:11', 'ingo',
'classes/@mfir/private/', '39b1930028da33d6bf793e8c01fde1bd'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/
private/mklowpass.m', 'mklowpass', '1.4', '2007/10/24', '10:55:11', 'ingo', 'classes/
@mfir/private/', '3f93a0a10a6e67a069a920db60318642'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/
private/parseFilterParams.m', 'parseFilterParams', '1.3', '2008/02/11', '12:31:11',
'mauro', 'classes/@mfir/private/', '811ddae1e3c332f9d58a68b5f090f99a'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/
private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:07', 'hewitson',
'classes/@mfir/private/', 'c8d5a2fab154101828c2d48628aade2b'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/
redesign.m', 'redesign', '1.4', '2008/02/12', '22:39:49', 'hewitson', 'classes/
@mfir/', 'd1b35155da13601889a762e276198da8'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/resp.
m', 'resp', '1.11', '2008/02/18', '18:08:03', 'mauro', 'classes/@mfir/',
'5b7a37259dd44f9d2421b8a8752aad50'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/save.
m', 'save', '1.5', '2008/02/15', '16:27:16', 'mauro', 'classes/@mfir/',
'a6004816a5af713dacbc9bfdeedd7291'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/set.
m', 'set', '1.7', '2008/02/13', '18:33:19', 'ingo', 'classes/@mfir/',
'2e65fcbfa14c2ec59e7f20a9d3a6c8dc'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/
string.m', 'string', '1.4', '2008/02/12', '22:39:49', 'hewitson', 'classes/@mfir/',
'd9ebdb92e2ecd6376a4f048be9505ef3'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/
subsasgn.m', 'subsasgn', '1.2', '2008/02/12', '22:39:49', 'hewitson', 'classes/
@mfir/', 'a1e55f06a8459cbc1622c80fcbc5663b'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@mfir/
subsref.m', 'subsref', '1.3', '2008/02/12', '22:39:49', 'hewitson', 'classes/
@mfir/', '42a58fea8d6e102e1bc8424c0493ff9b'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/char.
m', 'char', '1.5', '2008/02/12', '22:46:55', 'hewitson', 'classes/@miir/',
'3bcb087324efd651f9ed2a726298c56f'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/
display.m', 'display', '1.11', '2008/02/12', '22:46:55', 'hewitson', 'classes/
@miir/', '6dc2c8b7e8a627cbfdf3185ae5c47dec'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/eq.
m', 'eq', '1.5', '2008/02/18', '14:41:13', 'ingo', 'classes/@miir/',
'2b0e90ac1d471f40543e6e75dc37db12'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/get.
m', 'get', '1.6', '2008/02/11', '17:28:45', 'ingo', 'classes/@miir/',
'dbb2f3a15cdd223140b938f26c0531a2'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/
isfield.m', 'isfield', '1.4', '2008/02/15', '16:27:16', 'mauro', 'classes/@miir/',
'cc749a9d384ea821958d8ed822d353c8'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/
isvalid.m', 'isvalid', '1.2', '2008/02/15', '17:36:45', 'ingo', 'classes/@miir/',
'523dd2944b4870c44e2aa957d718a6ab'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/miir.
```

```
m', 'miir', '1.28', '2008/02/13', '17:31:11', 'hewitson', 'classes/@',
'1ed1e03b350a7189e56400868cc87bde'; ...
0170        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/ne.
m', 'ne', '1.4', '2008/02/18', '15:12:58', 'ingo', 'classes/@miir/',
'50e47e7037d169cb1ea5fd15920646d0'; ...
0171        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/
private/filload.m', 'filload', '1.4', '2007/11/11', '16:50:38', 'hewitson', 'classes/
@miir/private/', '479242466dc008db229041e4db07d2e2'; ...
0172        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/
private/mfiltload.m', 'mfiltload', '1.2', '2007/10/24', '10:55:11', 'ingo', 'classes/
@miir/private/', '4fb746f5ec5705bfe638372c3fc38ac3'; ...
0173        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/
private/mkbandpass.m', 'mkbandpass', '1.5', '2007/10/24', '10:55:11', 'ingo',
'classes/@miir/private/', '0d7e3d367260259e30cf3a18c189ca91'; ...
0174        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/
private/mkbandreject.m', 'mkbandreject', '1.7', '2008/01/02', '17:54:22', 'ingo',
'classes/@miir/private/', 'bc2284a8f6fdf7cb623a977ac1fcee0b'; ...
0175        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/
private/mkhighpass.m', 'mkhighpass', '1.5', '2007/10/24', '10:55:11', 'ingo',
'classes/@miir/private/', 'fb35fb0e807505e50f3bb832e3e560eb'; ...
0176        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/
private/mklowpass.m', 'mklowpass', '1.5', '2007/10/24', '10:55:11', 'ingo', 'classes/
@miir/private/', '913ec362130d24143fe3849975dd2287'; ...
0177        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/
private/parseFilterParams.m', 'parseFilterParams', '1.6', '2008/02/11', '11:38:40',
'mauro', 'classes/@miir/private/', '2ab311a119baee7635d3ed912f43d833'; ...
0178        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/
private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:08', 'hewitson',
'classes/@miir/private/', '0d2b03c20991c32ad109cf4a62d338b4'; ...
0179        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/
redesign.m', 'redesign', '1.9', '2008/02/12', '22:46:55', 'hewitson', 'classes/
@miir/', '4977c42dd5a33e37ee9c76105814bbdf'; ...
0180        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/resp.
m', 'resp', '1.9', '2008/02/18', '18:08:03', 'mauro', 'classes/@miir/',
'0731f5379092d79a0bd58bc77b03b079'; ...
0181        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/save.
m', 'save', '1.4', '2008/02/15', '16:27:16', 'mauro', 'classes/@miir/',
'25397edc2cada5f1a7539291f16467bb'; ...
0182        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/set.
m', 'set', '1.10', '2008/02/13', '18:33:19', 'ingo', 'classes/@miir/',
'efafa7a0986cbbb480e641ef35911923'; ...
0183        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/
string.m', 'string', '1.4', '2008/02/12', '22:46:55', 'hewitson', 'classes/@miir/',
'13bd7e54e072483b61999f763fcd2da5'; ...
0184        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/
subsasgn.m', 'subsasgn', '1.2', '2008/02/12', '22:46:55', 'hewitson', 'classes/
@miir/', '0c2da43024ae34fb238b66ef8da0617c'; ...
0185        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@miir/
subsref.m', 'subsref', '1.3', '2008/02/12', '22:46:55', 'hewitson', 'classes/
@miir/', 'ca1bd5b070454d489ff8913851ac5f5d'; ...
0186        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/cat.
m', 'cat', '1.4', '2008/02/13', '17:33:07', 'mauro', 'classes/@param/',
'063475b60fb43eb0ab33e7d1e8eeaca3'; ...
```

```
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/
char.m', 'char', '1.6', '2008/02/13', '17:33:38', 'mauro', 'classes/@param/',
'71dd30688084fe758119dd6b2f690a67'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/
display.m', 'display', '1.21', '2008/02/13', '17:33:38', 'mauro', 'classes/@param/',
'2ba5a0f765ce233e736141a87a24022e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/eq.
m', 'eq', '1.7', '2008/02/18', '14:41:13', 'ingo', 'classes/@param/',
'17005c51b1a251d47598b8f6cdcd4034'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/get.
m', 'get', '1.5', '2008/02/11', '17:28:45', 'ingo', 'classes/@param/',
'6142434450376378efa58bbc4c24be0e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/
isfield.m', 'isfield', '1.3', '2008/02/13', '17:33:38', 'mauro', 'classes/@param/',
'6876c4c849e7d4bf7226a75a41104ce6'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/
isvalid.m', 'isvalid', '1.1', '2008/02/14', '13:48:33', 'ingo', 'classes/@param/',
'6299e308c1adae408dcc4a0611d10e8d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/mux.
m', 'mux', '1.4', '2008/02/13', '17:33:38', 'mauro', 'classes/@param/',
'8a4edf9d22e1f3b6b31b7a4a3acf8cc7'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/ne.
m', 'ne', '1.4', '2008/02/18', '15:12:58', 'ingo', 'classes/@param/',
'a710e68d4a03ed7cf4168c895b5513d9'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/
param.m', 'param', '1.19', '2008/02/13', '17:33:38', 'mauro', 'classes/@',
'b7910b88030b7cfe873ab06f8fb3d3b2'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/
private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:08', 'hewitson',
'classes/@param/private/', '20ba8f9605e961c03e5a93a7ad0417d0'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/
save.m', 'save', '1.3', '2008/02/13', '17:33:38', 'mauro', 'classes/@param/',
'00d9f9051f209e17a3ec582592c233c0'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/set.
m', 'set', '1.9', '2008/02/12', '15:20:32', 'ingo', 'classes/@param/',
'5324cdf02f8f166656ffffff52ae5d74'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/
string.m', 'string', '1.1', '2008/02/15', '17:38:59', 'ingo', 'classes/@param/',
'aa60a6ee01e3b71382ee5b5004e5235a'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/
subsasgn.m', 'subsasgn', '1.5', '2008/02/13', '17:33:38', 'mauro', 'classes/
@param/', '6cf925a5d7e26fc6cc1f1bb2d841dbe6'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@param/
subsref.m', 'subsref', '1.8', '2008/02/13', '17:33:38', 'mauro', 'classes/@param/',
'1e75ec43012005e6deef43cd81c8837a'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
append.m', 'append', '1.8', '2008/02/14', '08:29:09', 'mauro', 'classes/@plist/',
'9b459bee939cc60bb3f186a06649d5d1'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
char.m', 'char', '1.13', '2008/02/14', '08:29:09', 'mauro', 'classes/@plist/',
'c5273a3fc442c8698fc4d5c11db0995c'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
combine.m', 'combine', '1.8', '2008/02/14', '08:29:09', 'mauro', 'classes/@plist/',
```

```
'f1f1d9bfcce86a38193ef95ae3dcf99c'; ...
0205         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
display.m', 'display', '1.5', '2008/02/14', '08:29:09', 'mauro', 'classes/@plist/',
'611532d036128cbcac129cca4fc132f4'; ...
0206         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/eq.
m', 'eq', '1.9', '2008/02/18', '14:41:13', 'ingo', 'classes/@plist/',
'6f77635d9625fa5f1be0ba53598ded6c'; ...
0207         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
find.m', 'find', '1.6', '2008/02/14', '08:29:09', 'mauro', 'classes/@plist/',
'31b42afc973e86039d6a6c7a62de4875'; ...
0208         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/get.
m', 'get', '1.5', '2008/02/11', '17:28:45', 'ingo', 'classes/@plist/',
'c491dc2ed3250faab4983f7e9752532e'; ...
0209         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
isfield.m', 'isfield', '1.4', '2008/02/14', '08:29:09', 'mauro', 'classes/@plist/',
'cf97973afbdc59e30e7c345277211ef5'; ...
0210         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
isparam.m', 'isparam', '1.3', '2008/02/14', '08:29:09', 'mauro', 'classes/@plist/',
'e60cde5370475c3fb9b6bc72726d7d8e'; ...
0211         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
isvalid.m', 'isvalid', '1.1', '2008/02/14', '13:48:34', 'ingo', 'classes/@plist/',
'686bf0e2edfa6090101aa7d92d171303'; ...
0212         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/ne.
m', 'ne', '1.5', '2008/02/18', '15:12:58', 'ingo', 'classes/@plist/',
'362b109f03d61a5d6a792232be4f4bfb'; ...
0213         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
nparams.m', 'nparams', '1.4', '2008/02/14', '08:29:09', 'mauro', 'classes/@plist/',
'bfaabdf14656a526636b7ffb02217f38'; ...
0214         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
plist.m', 'plist', '1.21', '2008/02/18', '09:32:32', 'ingo', 'classes/@',
'c66f17f72e62bc94c763b0563f074261'; ...
0215         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:08', 'hewitson',
'classes/@plist/private/', '3d940c3a40b01ebe9acfa16d2e4b1217'; ...
0216         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
pset.m', 'pset', '1.5', '2008/02/14', '08:29:09', 'mauro', 'classes/@plist/',
'6f3a4baf011318c4a75c8543b0b7d0ab'; ...
0217         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
pzmresp.m', 'pzmresp', '1.4', '2008/02/14', '08:29:09', 'mauro', 'classes/@plist/',
'512dffcd210357afb86c47bae762f58f'; ...
0218         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
remove.m', 'remove', '1.5', '2008/02/14', '08:29:09', 'mauro', 'classes/@plist/',
'31bf110b497f1507250b45a2328f3070'; ...
0219         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
resp.m', 'resp', '1.5', '2008/02/14', '14:31:38', 'mauro', 'classes/@plist/',
'c10433ec3bd592fecb0690451ea19ec7'; ...
0220         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
save.m', 'save', '1.3', '2008/02/14', '08:29:09', 'mauro', 'classes/@plist/',
'436480057b2c5bc46bb3b2f34e7ce933'; ...
0221         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/set.
m', 'set', '1.13', '2008/02/13', '18:33:19', 'ingo', 'classes/@plist/',
'52d17b8d45662c0f4853470085b853aa'; ...
0222         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
```

```
string.m', 'string', '1.11', '2008/02/15', '17:38:59', 'ingo', 'classes/@plist/',
'1340bcce7249425c735b481281f79f46'; ...
0223         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
subsasgn.m', 'subsasgn', '1.7', '2008/02/14', '08:29:09', 'mauro', 'classes/
@plist/', 'fa35bd52ea4428287799bbce9849c305'; ...
0224         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@plist/
subsref.m', 'subsref', '1.14', '2008/02/14', '08:29:09', 'mauro', 'classes/@plist/',
'b8f20d1aa4ece5923e43db90c7b7fc27'; ...
0225         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/cat.
m', 'cat', '1.4', '2008/02/12', '21:38:23', 'hewitson', 'classes/@pole/',
'7f66ec7b8aeee72c77ebf708035d5455'; ...
0226         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/char.
m', 'char', '1.4', '2008/02/12', '21:38:23', 'hewitson', 'classes/@pole/',
'd408827cd87d9f672205130694188dce'; ...
0227         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/
cp2iir.m', 'cp2iir', '1.5', '2008/02/12', '21:38:23', 'hewitson', 'classes/@pole/',
'85f599b4e1b9000b8507dfe9038c13dc'; ...
0228         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/
display.m', 'display', '1.5', '2008/02/12', '21:38:23', 'hewitson', 'classes/
@pole/', '86437edef6d01d1f5d47823a085a8826'; ...
0229         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/eq.
m', 'eq', '1.5', '2008/02/18', '14:41:13', 'ingo', 'classes/@pole/',
'7a595083d28543b679f54db444137e83'; ...
0230         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/get.
m', 'get', '1.6', '2008/02/11', '17:28:45', 'ingo', 'classes/@pole/',
'a23189b6986b96d7f7d5b67919bc0d52'; ...
0231         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/
isfield.m', 'isfield', '1.4', '2008/02/15', '16:27:16', 'mauro', 'classes/@pole/',
'68701ed6c10f972910691715c26d638d'; ...
0232         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/
isvalid.m', 'isvalid', '1.1', '2008/02/14', '13:48:34', 'ingo', 'classes/@pole/',
'fdfeac90aa9d7320fb2121c63b865db0'; ...
0233         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/ne.
m', 'ne', '1.4', '2008/02/18', '15:12:58', 'ingo', 'classes/@pole/',
'dc8c02445627ae21cfe8e170209209bb'; ...
0234         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/pole.
m', 'pole', '1.20', '2008/02/17', '11:38:51', 'hewitson', 'classes/@',
'e7d84bfb871a7269f98cc609c66e9980'; ...
0235         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/
private/pfq2ri.m', 'pfq2ri', '1.10', '2008/02/13', '10:13:54', 'ingo', 'classes/
@pole/private/', '6ce281f816e68522c5a9a1eab545d57c'; ...
0236         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/
private/pri2fq.m', 'pri2fq', '1.3', '2008/01/17', '16:00:06', 'anneke', 'classes/
@pole/private/', '2bb9270d4a953110924766641d979c4f'; ...
0237         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/
private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:08', 'hewitson',
'classes/@pole/private/', 'ce85d3f372e69ff29ecae0c57bdeeb8a'; ...
0238         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/resp.
m', 'resp', '1.7', '2008/02/14', '14:31:38', 'mauro', 'classes/@pole/',
'e2fc4eedca1e43937b9adb8c9872577c'; ...
0239         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/
rp2iir.m', 'rp2iir', '1.5', '2008/02/12', '21:38:23', 'hewitson', 'classes/@pole/',
'7d9d3eb388f6244dcc9afcedc1509930'; ...
```

```
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/save.m', 'save', '1.4', '2008/02/15', '16:27:16', 'mauro', 'classes/@pole/', '1f52819c052eb334b53295172d985ee7'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/set.m', 'set', '1.10', '2008/02/16', '22:22:42', 'hewitson', 'classes/@pole/', 'bed240b7ef864a4adf32ffff730dbfc4'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/string.m', 'string', '1.7', '2008/02/15', '17:38:59', 'ingo', 'classes/@pole/', '75a873a1b1f03b3e2e3a87f89602d974'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/subsasgn.m', 'subsasgn', '1.3', '2008/02/13', '10:13:54', 'ingo', 'classes/@pole/', '9bcc05a764b5698fab21e8949564a487'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pole/subsref.m', 'subsref', '1.3', '2008/02/12', '21:38:23', 'hewitson', 'classes/@pole/', '3582ccd0298fc70914abeffb94b8fc36'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@provenance/char.m', 'char', '1.6', '2008/02/13', '17:11:03', 'mauro', 'classes/@provenance/', 'dd45d600dd2617cf618d91e4756e677f'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@provenance/display.m', 'display', '1.9', '2008/02/13', '17:11:03', 'mauro', 'classes/@provenance/', '23f0d1f3aa4be027f21bf3ddaface9a2'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@provenance/eq.m', 'eq', '1.6', '2008/02/18', '14:41:13', 'ingo', 'classes/@provenance/', '25d175ba08fbcb0e821b8f71043f9062'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@provenance/get.m', 'get', '1.4', '2008/02/11', '17:28:45', 'ingo', 'classes/@provenance/', '88dff06ff9aa4e5533155eddc6cdf505'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@provenance/isfield.m', 'isfield', '1.3', '2008/02/13', '17:11:03', 'mauro', 'classes/@provenance/', '0e30507ff2dfbaad17f5892c66b2c1c8'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@provenance/isvalid.m', 'isvalid', '1.1', '2008/02/14', '13:48:34', 'ingo', 'classes/@provenance/', '4e6514bd19f4aad3ad055c048434ff93'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@provenance/ne.m', 'ne', '1.4', '2008/02/18', '15:12:58', 'ingo', 'classes/@provenance/', '071cd3b2d1ba8521571f5931c477c360'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@provenance/private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:09', 'hewitson', 'classes/@provenance/private/', '9e776926b46a37e69bfbe6097644103b'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@provenance/provenance.m', 'provenance', '1.19', '2008/02/15', '17:35:55', 'ingo', 'classes/@', '5d0d631c06a698557d1622c7c1bfacbe'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@provenance/save.m', 'save', '1.3', '2008/02/13', '17:11:03', 'mauro', 'classes/@provenance/', '2456be2cf7cafa3eb5b1aaa6b249b5be'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@provenance/set.m', 'set', '1.5', '2008/02/12', '15:20:33', 'ingo', 'classes/@provenance/', '8487b15ec37d574062cf54440f34d93a'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@provenance/string.m', 'string', '1.1', '2008/02/15', '17:38:59', 'ingo', 'classes/@provenance/', 'c0c6e130a9e838e824a6b1e0d6c799c7'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@provenance/subsasgn.m', 'subsasgn', '1.2', '2008/02/13', '17:11:04', 'mauro',
```

```
'classes/@provenance/', '4c6690adfe359d14e40813734998b4b7'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/
@provenance/subsref.m', 'subsref', '1.4', '2008/02/13', '17:11:04', 'mauro',
'classes/@provenance/', 'c3800ede22277dfaa30bd25fd337902e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
char.m', 'char', '1.4', '2008/02/12', '20:07:37', 'hewitson', 'classes/@pzmodel/',
'fdb4682c573eb178e0cf74e13c3c2c45'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
display.m', 'display', '1.6', '2008/02/12', '20:07:37', 'hewitson', 'classes/
@pzmodel/', '541041f7fcb6a10dfd5dd1de0d255be6'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
eq.m', 'eq', '1.7', '2008/02/18', '14:41:13', 'ingo', 'classes/@pzmodel/',
'a35bffde50b2155f4300aa0b4e1f1121'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
get.m', 'get', '1.4', '2008/02/11', '17:28:45', 'ingo', 'classes/@pzmodel/',
'033da57b10f7d1874af873c8b1d59bfc'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
getlowerFreq.m', 'getlowerFreq', '1.4', '2008/02/12', '20:07:37', 'hewitson',
'classes/@pzmodel/', '6dbc66108aa88e176b07fbe09e1d5e8b'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
getupperFreq.m', 'getupperFreq', '1.4', '2008/02/12', '20:07:37', 'hewitson',
'classes/@pzmodel/', '81eeb15b5c8c797cfcde811db7b8264b'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
isfield.m', 'isfield', '1.4', '2008/02/15', '16:27:16', 'mauro', 'classes/
@pzmodel/', 'e76a7a8293a9ffa205a22b0deb96c68b'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
isvalid.m', 'isvalid', '1.1', '2008/02/14', '13:48:34', 'ingo', 'classes/@pzmodel/',
'a557174b7677ee888f51d82e98f594de'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
ne.m', 'ne', '1.5', '2008/02/18', '15:12:58', 'ingo', 'classes/@pzmodel/',
'bb22537da6aa5dc39c1f5fc518102707'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
private/abcascade.m', 'abcascade', '1.1', '2007/04/03', '11:34:23', 'hewitson',
'classes/@pzmodel/private/', '1b885675a4b790563050ccf13279856a'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
private/pzm2ab.m', 'pzm2ab', '1.4', '2007/09/03', '09:27:04', 'hewitson', 'classes/
@pzmodel/private/', 'f1be32574fcab3fbc5b9622c0a8f997e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:08', 'hewitson',
'classes/@pzmodel/private/', '0a66b509bb20ce468dd15e1fe4820659'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
pzmodel.m', 'pzmodel', '1.18', '2008/02/14', '15:22:09', 'hewitson', 'classes/@',
'182c61f21b8fcb91045db8c3351008d3'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
resp.m', 'resp', '1.12', '2008/02/14', '15:22:09', 'hewitson', 'classes/@pzmodel/',
'58d5c8a917bf52fc942b712512c94fd8'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
save.m', 'save', '1.4', '2008/02/15', '16:27:16', 'mauro', 'classes/@pzmodel/',
'5e4951f2c31af0863addc841cdbec448'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
set.m', 'set', '1.6', '2008/02/13', '18:33:19', 'ingo', 'classes/@pzmodel/',
'aade709e14a83ce9b4487a4bdb4a146d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
```

```
string.m', 'string', '1.6', '2008/02/12', '20:07:37', 'hewitson', 'classes/
@pzmodel/', '87aa5acd849e65602ae577ea1cf72eb2'; ...
0276         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
subsasgn.m', 'subsasgn', '1.2', '2008/02/12', '20:07:37', 'hewitson', 'classes/
@pzmodel/', 'edb8cd0f830c02a5f75c312996b0e8bc'; ...
0277         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
subsref.m', 'subsref', '1.3', '2008/02/12', '20:07:37', 'hewitson', 'classes/
@pzmodel/', '3c83bdb46fd2f53d3c85d7c0f15aa91a'; ...
0278         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
tomfir.m', 'tomfir', '1.3', '2008/02/12', '20:07:37', 'hewitson', 'classes/
@pzmodel/', '555bd284e5295aac742ea0bb8032cd55'; ...
0279         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@pzmodel/
tomiir.m', 'tomiir', '1.6', '2008/02/12', '20:07:37', 'hewitson', 'classes/
@pzmodel/', '5e54abc5a4a962c97f4b03825b0fea65'; ...
0280         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
char.m', 'char', '1.4', '2008/02/13', '13:46:07', 'mauro', 'classes/@specwin/',
'de8458393131d4e71b68ad97cd0a76ad'; ...
0281         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
display.m', 'display', '1.7', '2008/02/13', '13:46:07', 'mauro', 'classes/
@specwin/', 'dc4b2414f0c6c5141e3cc5a6f1efbb8b'; ...
0282         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
eq.m', 'eq', '1.5', '2008/02/18', '14:41:13', 'ingo', 'classes/@specwin/',
'd844b49b73396319bd09a28adfbbe31f'; ...
0283         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
get.m', 'get', '1.4', '2008/02/11', '17:28:45', 'ingo', 'classes/@specwin/',
'5ff3b175b441fd223522dadc33f036fd'; ...
0284         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
isfield.m', 'isfield', '1.3', '2008/02/13', '13:46:07', 'mauro', 'classes/
@specwin/', '14cb461d6c7cdbb5bf8d76d3773a827e'; ...
0285         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
isvalid.m', 'isvalid', '1.2', '2008/02/17', '18:45:56', 'hewitson', 'classes/
@specwin/', '43c8595fae021134c2532fdacd77a6d1'; ...
0286         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
ne.m', 'ne', '1.4', '2008/02/18', '15:12:58', 'ingo', 'classes/@specwin/',
'a158fb879fc48d49d9be0040b782452b'; ...
0287         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
plot.m', 'plot', '1.6', '2008/02/13', '13:46:07', 'mauro', 'classes/@specwin/',
'960cc5e44f1b0aea6acb696ebf3524a9'; ...
0288         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/get_window.m', 'get_window', '1.5', '2008/02/13', '18:38:02', 'ingo',
'classes/@specwin/private/', '6e39a94f6bdd61ea692c1d68ae969aa1'; ...
0289         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/kaiser_alpha.m', 'kaiser_alpha', '1.1', '2007/05/19', '09:04:59',
'hewitson', 'classes/@specwin/private/', '2545b51d7993c9db15f389f6fa59d5be'; ...
0290         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/kaiser_flatness.m', 'kaiser_flatness', '1.1', '2007/05/19', '09:04:59',
'hewitson', 'classes/@specwin/private/', 'b6b3dec2f5ace41e2a25650a2254011b'; ...
0291         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/kaiser_nenbw.m', 'kaiser_nenbw', '1.1', '2007/05/19', '09:04:59',
'hewitson', 'classes/@specwin/private/', 'fa194454fc803c7fed9a94aa4d80c483'; ...
0292         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/kaiser_rov.m', 'kaiser_rov', '1.1', '2007/05/19', '09:04:59', 'hewitson',
'classes/@specwin/private/', 'b98a36c787f2b09dfb01522e684c4cb0'; ...
```

```
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/kaiser_w3db.m', 'kaiser_w3db', '1.1', '2007/05/19', '09:04:59', 'hewitson', 'classes/@specwin/private/', '23f7d810dcd9fe19d23b18e6a518a4fd'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:08', 'hewitson', 'classes/@specwin/private/', 'fbda45570ddb5b3e2ae334ee3576aaca'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_BH92.m', 'win_BH92', '1.2', '2008/02/13', '18:38:02', 'ingo', 'classes/@specwin/private/', '66877f6b61566aa034869fa566d26286'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_Bartlett.m', 'win_Bartlett', '1.2', '2008/02/13', '18:38:02', 'ingo', 'classes/@specwin/private/', '97753d2d220e46fe9e8bbacac294b33f'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_FTHP.m', 'win_FTHP', '1.2', '2008/02/13', '18:38:02', 'ingo', 'classes/@specwin/private/', '7095af97c9119c2955dfeb22b694c5d4'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_FTNI.m', 'win_FTNI', '1.2', '2008/02/13', '18:38:02', 'ingo', 'classes/@specwin/private/', '8673cf9067e45453957d42739f2b2625'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_FTSRS.m', 'win_FTSRS', '1.2', '2008/02/13', '18:38:02', 'ingo', 'classes/@specwin/private/', '62a090a5f807b37c84f451cfd0cb52d8'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_HFT116D.m', 'win_HFT116D', '1.2', '2008/02/13', '18:38:02', 'ingo', 'classes/@specwin/private/', 'a7dd3d7bc1b3c67d45ebac0987081f65'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_HFT144D.m', 'win_HFT144D', '1.3', '2008/02/17', '18:46:31', 'hewitson', 'classes/@specwin/private/', 'b4110c800203d7ea6fba44f894e7ddd0'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_HFT169D.m', 'win_HFT169D', '1.2', '2008/02/13', '18:38:02', 'ingo', 'classes/@specwin/private/', '3368561750e82dea5e62fce32ec56666'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_HFT196D.m', 'win_HFT196D', '1.2', '2008/02/13', '18:38:02', 'ingo', 'classes/@specwin/private/', 'dc4e5d7fed4e5355787f8cda0888c3f8'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_HFT223D.m', 'win_HFT223D', '1.2', '2008/02/13', '18:38:02', 'ingo', 'classes/@specwin/private/', 'ba7482f72af4d810cf17e1af1495a543'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_HFT248D.m', 'win_HFT248D', '1.3', '2008/02/17', '18:46:31', 'hewitson', 'classes/@specwin/private/', '7b2d84ccccfd3114e13a82bcc5dff23a'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_HFT70.m', 'win_HFT70', '1.2', '2008/02/13', '18:38:02', 'ingo', 'classes/@specwin/private/', 'a4808876a8cd5c0f00ff26db5952d7b9'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_HFT90D.m', 'win_HFT90D', '1.2', '2008/02/13', '18:38:02', 'ingo', 'classes/@specwin/private/', '6c0a3ece268cedc3a24a15dfe8a22f25'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_HFT95.m', 'win_HFT95', '1.2', '2008/02/13', '18:38:02', 'ingo', 'classes/@specwin/private/', '45cac82b7ed75fb526a8fb0586dafc88'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_Hamming.m', 'win_Hamming', '1.2', '2008/02/13', '18:38:02', 'ingo', 'classes/@specwin/private/', '158cfe6a3727a9d74856c1c656fcac3d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/private/win_Hanning.m', 'win_Hanning', '1.2', '2008/02/13', '18:38:02', 'ingo',
```

```
'classes/@specwin/private/', '1cc186aa01ecb946b1440b6b6d847546'; ...
0311        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/win_Nuttall3.m', 'win_Nuttall3', '1.2', '2008/02/13', '18:38:02', 'ingo',
'classes/@specwin/private/', 'd365c3cd78ec5ed6af0c0a81ac55a755'; ...
0312        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/win_Nuttall3a.m', 'win_Nuttall3a', '1.2', '2008/02/13', '18:38:02', 'ingo',
'classes/@specwin/private/', '2e370c67a0d02634cb8c2635ba281c79'; ...
0313        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/win_Nuttall3b.m', 'win_Nuttall3b', '1.2', '2008/02/13', '18:38:02', 'ingo',
'classes/@specwin/private/', 'e1f52d3f0171561848079b8edbc07b38'; ...
0314        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/win_Nuttall4.m', 'win_Nuttall4', '1.2', '2008/02/13', '18:38:03', 'ingo',
'classes/@specwin/private/', 'ab5aa76bf84036d71b41e2734768b366'; ...
0315        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/win_Nuttall4a.m', 'win_Nuttall4a', '1.2', '2008/02/13', '18:38:02', 'ingo',
'classes/@specwin/private/', '56c6061a5ff10e66a235f3af75541962'; ...
0316        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/win_Nuttall4b.m', 'win_Nuttall4b', '1.2', '2008/02/13', '18:38:03', 'ingo',
'classes/@specwin/private/', 'ca703c4779328a76b6d3e5b9289cccd6'; ...
0317        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/win_Nuttall4c.m', 'win_Nuttall4c', '1.2', '2008/02/13', '18:38:03', 'ingo',
'classes/@specwin/private/', 'a7a7d7724d89560d2a410f4f8137c32f'; ...
0318        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/win_Rectangular.m', 'win_Rectangular', '1.2', '2008/02/13', '18:38:03',
'ingo', 'classes/@specwin/private/', '6a88c1aabf055b27f55c6b96068f054f'; ...
0319        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/win_SFT3F.m', 'win_SFT3F', '1.2', '2008/02/13', '18:38:03', 'ingo', 'classes/
@specwin/private/', 'af2ecf2ff75d1085b1c8c5a9cba04875'; ...
0320        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/win_SFT3M.m', 'win_SFT3M', '1.2', '2008/02/13', '18:38:03', 'ingo', 'classes/
@specwin/private/', '9bffaa009f081b382e3b824c7ce36fc3'; ...
0321        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/win_SFT4F.m', 'win_SFT4F', '1.2', '2008/02/13', '18:38:03', 'ingo', 'classes/
@specwin/private/', 'f64d9103d4929459ba73264a38ae8bd6'; ...
0322        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/win_SFT4M.m', 'win_SFT4M', '1.2', '2008/02/13', '18:38:03', 'ingo', 'classes/
@specwin/private/', '05e03933ba09f59556dbd038e0384861'; ...
0323        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/win_SFT5F.m', 'win_SFT5F', '1.2', '2008/02/13', '18:38:03', 'ingo', 'classes/
@specwin/private/', '44fa73849b308b72b2c7aab41fc68cf9'; ...
0324        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/win_SFT5M.m', 'win_SFT5M', '1.2', '2008/02/13', '18:38:03', 'ingo', 'classes/
@specwin/private/', 'ee51dca74d48ee7c1a0ee4ff7675a6ad'; ...
0325        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
private/win_Welch.m', 'win_Welch', '1.2', '2008/02/13', '18:38:03', 'ingo', 'classes/
@specwin/private/', 'fe8ae3067f00bec3308d3597c7b06278'; ...
0326        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
save.m', 'save', '1.3', '2008/02/13', '13:46:07', 'mauro', 'classes/@specwin/',
'76ddaf6b17e5218adc6af5ab41411870'; ...
0327        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
set.m', 'set', '1.6', '2008/02/13', '18:38:03', 'ingo', 'classes/@specwin/',
'b7c836eea4f1196bb4bfd52f1aa6ee49'; ...
0328        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
```

```
specwin.m', 'specwin', '1.24', '2008/02/18', '13:36:29', 'ingo', 'classes/@',
'a8f453f6e5a779c54beff725ea6c1431'; ...
0329         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
string.m', 'string', '1.4', '2008/02/13', '13:46:07', 'mauro', 'classes/@specwin/',
'21d2a2909558b5580ba0989cf91c06d5'; ...
0330         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
subsasgn.m', 'subsasgn', '1.3', '2008/02/13', '13:46:07', 'mauro', 'classes/
@specwin/', 'b87e97a403cd9b7956069ed1611ff9e8'; ...
0331         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@specwin/
subsref.m', 'subsref', '1.4', '2008/02/13', '13:46:07', 'mauro', 'classes/
@specwin/', '5d090181c19917a3a7578bfc71df40b0'; ...
0332         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/char.
m', 'char', '1.5', '2008/02/13', '16:31:26', 'mauro', 'classes/@time/',
'c2ecace1786be2852ef77abd033e180d'; ...
0333         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/
display.m', 'display', '1.9', '2008/02/13', '16:31:26', 'mauro', 'classes/@time/',
'5d6249755a9412c29c62bb702479021c'; ...
0334         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/eq.
m', 'eq', '1.7', '2008/02/18', '14:41:13', 'ingo', 'classes/@time/',
'2ddd86fbc10328006256e34f346a9745'; ...
0335         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/
format.m', 'format', '1.7', '2008/02/13', '16:31:26', 'mauro', 'classes/@time/',
'038c43cb971d284a5972f8fcf5ad04e8'; ...
0336         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/get.
m', 'get', '1.10', '2008/02/11', '17:28:45', 'ingo', 'classes/@time/',
'8a8c8cca59fbe1a811da9fae381b32ea'; ...
0337         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/
isfield.m', 'isfield', '1.4', '2008/02/13', '16:47:35', 'mauro', 'classes/@time/',
'e9f278be82856a6740a0289e1fdee5b6'; ...
0338         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/
isvalid.m', 'isvalid', '1.1', '2008/02/14', '13:48:34', 'ingo', 'classes/@time/',
'6fc0086f8e0c9c589bcfcdce2afb25ed'; ...
0339         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/
minus.m', 'minus', '1.6', '2008/02/15', '16:26:08', 'mauro', 'classes/@time/',
'a75be4c3d479728e6fb9f3f873185c03'; ...
0340         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/ne.
m', 'ne', '1.4', '2008/02/18', '15:12:58', 'ingo', 'classes/@time/',
'f9473354e1d7fda6e7178702ca3ee3a5'; ...
0341         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/plus.
m', 'plus', '1.7', '2008/02/15', '16:26:08', 'mauro', 'classes/@time/',
'a442428f03c8f534f2c945bf45d1f553'; ...
0342         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/
private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:07', 'hewitson',
'classes/@time/private/', '63c36a60bd2e379ff0b9de8c2d3d7841'; ...
0343         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/save.
m', 'save', '1.3', '2008/02/13', '16:31:26', 'mauro', 'classes/@time/',
'f11c80a424e55e02d6f471fd3bf9082c'; ...
0344         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/set.
m', 'set', '1.14', '2008/02/15', '17:35:26', 'ingo', 'classes/@time/',
'ca9f5bcf6e67b01321274ba85678ca3e'; ...
0345         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/
string.m', 'string', '1.4', '2008/02/19', '12:10:23', 'ingo', 'classes/@time/',
'23f1e37023f5c0beeabbdca3f9f4d3bc'; ...
```

```
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/
subsasgn.m', 'subsasgn', '1.2', '2008/02/13', '16:31:26', 'mauro', 'classes/@time/',
'fd441d7de6e94cc518617036288eb8e2'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/
subsref.m', 'subsref', '1.8', '2008/02/13', '16:31:26', 'mauro', 'classes/@time/',
'930fe6152c860d6a0138bb96bc8e25ba'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@time/time.
m', 'time', '1.20', '2008/02/15', '17:34:55', 'ingo', 'classes/@',
'd69fb5b0973218ae9f70910a3dbb1aef'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/
@timeformat/char.m', 'char', '1.3', '2008/02/13', '16:54:14', 'mauro', 'classes/
@timeformat/', '24ecbf76a778217d93ccb2cbd29767a7'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/
@timeformat/display.m', 'display', '1.5', '2008/02/13', '16:54:14', 'mauro',
'classes/@timeformat/', '9088a2e47fe1394de19a836eba7ac50d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/
@timeformat/eq.m', 'eq', '1.5', '2008/02/18', '14:41:13', 'ingo', 'classes/
@timeformat/', '5e4441881dead576b1c42e3096f8d78f'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/
@timeformat/get.m', 'get', '1.1', '2008/02/11', '17:28:45', 'ingo', 'classes/
@timeformat/', '53dfed868a5521faa97655f41f85f7fa'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/
@timeformat/isfield.m', 'isfield', '1.3', '2008/02/13', '16:54:14', 'mauro',
'classes/@timeformat/', '529e31084d0cff95980204bc2099a0b7'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/
@timeformat/isvalid.m', 'isvalid', '1.1', '2008/02/14', '13:48:34', 'ingo', 'classes/
@timeformat/', 'd1669e75fc29dab9838fe3e1f12f5e22'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/
@timeformat/ne.m', 'ne', '1.6', '2008/02/18', '19:54:21', 'hewitson', 'classes/
@timeformat/', '99df0e469117e3b13e0f67afbc63dfe9'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/
@timeformat/private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:08',
'hewitson', 'classes/@timeformat/private/', '6484cb7706c5d49ab8a05b7d947ceefa'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/
@timeformat/save.m', 'save', '1.3', '2008/02/13', '16:54:14', 'mauro', 'classes/
@timeformat/', '8b6a9722750ba1c4eb23e601b62e3d86'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/
@timeformat/set.m', 'set', '1.4', '2008/02/12', '15:20:33', 'ingo', 'classes/
@timeformat/', '02c99e3812d97631e4bc4cfa28999c2c'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/
@timeformat/string.m', 'string', '1.1', '2008/02/15', '17:38:59', 'ingo', 'classes/
@timeformat/', '6942b7ae81a4997a2780ffcc34d510eb'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/
@timeformat/subsasgn.m', 'subsasgn', '1.2', '2008/02/13', '16:54:14', 'mauro',
'classes/@timeformat/', '10ff01bc9bbeaf17992f5f094571e6e3'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/
@timeformat/subsref.m', 'subsref', '1.3', '2008/02/13', '16:54:14', 'mauro',
'classes/@timeformat/', '71ee165f1667d081be4859ebde87fad8'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/
@timeformat/timeformat.m', 'timeformat', '1.8', '2008/02/13', '16:54:14', 'mauro',
'classes/@', '25a0579ec34156457d94ee21f99e50bd'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@timespan/
char.m', 'char', '1.3', '2008/02/13', '16:42:51', 'mauro', 'classes/@timespan/',
```

```
'88970580c9e7515076c04ac86cf598d1'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@timespan/
display.m', 'display', '1.6', '2008/02/13', '16:43:13', 'mauro', 'classes/
@timespan/', 'c90ec0c1e829aadbbaa3bcb45781d8f2'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@timespan/
eq.m', 'eq', '1.5', '2008/02/18', '14:41:13', 'ingo', 'classes/@timespan/',
'0684570df185ae1b4471777dd5fa8f46'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@timespan/
get.m', 'get', '1.1', '2008/02/11', '17:28:45', 'ingo', 'classes/@timespan/',
'f4e2a202e52e5c6353e968b964c06d7f'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@timespan/
isfield.m', 'isfield', '1.3', '2008/02/13', '16:43:14', 'mauro', 'classes/
@timespan/', 'c556b7c7a5375505b85bee96485a77c4'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@timespan/
isvalid.m', 'isvalid', '1.1', '2008/02/14', '13:48:34', 'ingo', 'classes/
@timespan/', 'ff3a8ddfcd58373e48f0a8a5510ffce3'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@timespan/
ne.m', 'ne', '1.4', '2008/02/18', '15:12:58', 'ingo', 'classes/@timespan/',
'df5445c497c41ccc906f166871336e2e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@timespan/
private/compute_interval.m', 'compute_interval', '1.3', '2008/02/13', '10:20:16',
'ingo', 'classes/@timespan/private/', 'ce3c0e72d798e18414af8c6b5b2b77f2'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@timespan/
private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:08', 'hewitson',
'classes/@timespan/private/', 'b39ff987f44413ff3e4e44f0eccf2eb4'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@timespan/
save.m', 'save', '1.3', '2008/02/13', '16:43:14', 'mauro', 'classes/@timespan/',
'3f9a799b4c76b6e8d62928fbda306c43'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@timespan/
set.m', 'set', '1.13', '2008/02/15', '17:35:26', 'ingo', 'classes/@timespan/',
'6f3f3a4aa694804331b93e9ab231400e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@timespan/
string.m', 'string', '1.1', '2008/02/15', '17:38:59', 'ingo', 'classes/@timespan/',
'41ceab4eec9f5f74967e7941d4fdd4a1'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@timespan/
subsasgn.m', 'subsasgn', '1.2', '2008/02/13', '16:43:14', 'mauro', 'classes/
@timespan/', '266f257ad5a37a42a728ca85a28155de'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@timespan/
subsref.m', 'subsref', '1.3', '2008/02/13', '16:43:14', 'mauro', 'classes/
@timespan/', 'f888a8942648667528291ee2705ac160'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@timespan/
timespan.m', 'timespan', '1.15', '2008/02/15', '17:34:02', 'ingo', 'classes/@',
'fc0addacbec0ba202d357f3952cc8e07'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
char.m', 'char', '1.4', '2008/02/13', '11:39:35', 'mauro', 'classes/@tsdata/',
'1e752f3efce7dc26668cfb5b591c4cc9'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
display.m', 'display', '1.10', '2008/02/13', '11:39:35', 'mauro', 'classes/
@tsdata/', '1e2cd64c923a9ca1454019881a751e48'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/eq.
m', 'eq', '1.9', '2008/02/18', '14:41:13', 'ingo', 'classes/@tsdata/',
'3181f43fd88704fda16e9ef3f8a9f739'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
```

```
get.m', 'get', '1.11', '2008/02/11', '17:28:46', 'ingo', 'classes/@tsdata/',
'53d59e627aad6bae2145b98b9b2454c1'; ...
0382        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
get_xy_values.m', 'get_xy_values', '1.7', '2008/02/13', '11:39:35', 'mauro',
'classes/@tsdata/', 'a9e8c49f2b44a22d6a344e14a1c614bc'; ...
0383        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
isfield.m', 'isfield', '1.4', '2008/02/15', '16:27:16', 'mauro', 'classes/@tsdata/',
'd882693a1c99a39daa743848e2a4c767'; ...
0384        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
isvalid.m', 'isvalid', '1.1', '2008/02/14', '13:48:34', 'ingo', 'classes/@tsdata/',
'dbe6fc4517717385f0abf18c0d5f5687'; ...
0385        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/ne.
m', 'ne', '1.4', '2008/02/18', '15:12:58', 'ingo', 'classes/@tsdata/',
'28ff2afb1bd0a935a6fc89f435c0e658'; ...
0386        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:09', 'hewitson',
'classes/@tsdata/private/', '44c2b3e9c7e656542fd0b2ca0d10c9a2'; ...
0387        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
reshapeT.m', 'reshapeT', '1.5', '2008/02/13', '11:39:35', 'mauro', 'classes/
@tsdata/', 'f06187c86d18a1bf6147dde2675f37e8'; ...
0388        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
save.m', 'save', '1.4', '2008/02/15', '16:27:16', 'mauro', 'classes/@tsdata/',
'5895493ff778e78093faa4abfd6fe010'; ...
0389        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
set.m', 'set', '1.14', '2008/02/12', '15:20:33', 'ingo', 'classes/@tsdata/',
'cceb3667bd64d0f6bbc995bf52447247'; ...
0390        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
setTime.m', 'setTime', '1.7', '2008/02/13', '12:12:49', 'mauro', 'classes/@tsdata/',
'3feed0f78d5f72fd38d7ad971dbcea52'; ...
0391        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
set_xy_axis.m', 'set_xy_axis', '1.5', '2008/02/13', '12:12:49', 'mauro', 'classes/
@tsdata/', '3a95eaa077e5ba4db92b7e3d29dbe379'; ...
0392        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
single_operation.m', 'single_operation', '1.12', '2008/02/13', '12:12:49', 'mauro',
'classes/@tsdata/', '5c3c4332c807a0fbede3d29ee048c19e'; ...
0393        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
subsasgn.m', 'subsasgn', '1.4', '2008/02/13', '12:12:49', 'mauro', 'classes/
@tsdata/', 'ac32a8f707c2a02ac1ed60ff2533e853'; ...
0394        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
subsref.m', 'subsref', '1.14', '2008/02/13', '12:12:49', 'mauro', 'classes/
@tsdata/', '6c785e317e6e58a340a9e28f1eb9c41f'; ...
0395        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@tsdata/
tsdata.m', 'tsdata', '1.24', '2008/02/13', '12:12:49', 'mauro', 'classes/@',
'd5b56ee3bfe9482aeef749d6afcd5a04'; ...
0396        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/
char.m', 'char', '1.4', '2008/02/13', '12:57:47', 'mauro', 'classes/@xydata/',
'6422b9eb6f537718b11ada68a4d43adc'; ...
0397        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/
display.m', 'display', '1.7', '2008/02/13', '12:57:47', 'mauro', 'classes/@xydata/',
'0ac5e804ce6e65650e30d3971e495208'; ...
0398        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/eq.
m', 'eq', '1.8', '2008/02/18', '14:41:13', 'ingo', 'classes/@xydata/',
'e9c9a59f974ef2252a0ea37673df5d8e'; ...
```

```
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/
get.m', 'get', '1.5', '2008/02/11', '17:28:46', 'ingo', 'classes/@xydata/',
'0e54072614e50e4f009fb32016c73ff8'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/
get_xy_values.m', 'get_xy_values', '1.4', '2008/02/13', '12:57:47', 'mauro',
'classes/@xydata/', '1741e34d0b706a3e7eb4294e1969e743'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/
isfield.m', 'isfield', '1.4', '2008/02/15', '16:27:16', 'mauro', 'classes/@xydata/',
'93af623b550ff23be7503aabcf1044aa'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/
isvalid.m', 'isvalid', '1.1', '2008/02/14', '13:48:34', 'ingo', 'classes/@xydata/',
'beeb522c8ddb981af71c63a296338b7d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/ne.
m', 'ne', '1.4', '2008/02/18', '15:12:58', 'ingo', 'classes/@xydata/',
'fb68f22881b016e95c99ec0a8a2ecd76'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/
private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:08', 'hewitson',
'classes/@xydata/private/', '4b321b39d12f8e94519bfb6bd216e064'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/
reshapeX.m', 'reshapeX', '1.5', '2008/02/13', '12:57:47', 'mauro', 'classes/
@xydata/', 'eceb58c8ef6ca5fc42b502f9f195f8d6'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/
save.m', 'save', '1.4', '2008/02/15', '16:27:16', 'mauro', 'classes/@xydata/',
'3c09a85d5680250b816e20d2364ddc0c'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/
set.m', 'set', '1.8', '2008/02/12', '15:20:33', 'ingo', 'classes/@xydata/',
'7d506ed7e969c2b3b60d23248e4697b3'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/
set_xy_axis.m', 'set_xy_axis', '1.4', '2008/02/13', '12:57:47', 'mauro', 'classes/
@xydata/', '3b9e7512fa830275b5743eaf9817f371'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/
single_operation.m', 'single_operation', '1.12', '2008/02/13', '12:57:47', 'mauro',
'classes/@xydata/', '59f581d15cca6a1956601f7f132d975d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/
subsasgn.m', 'subsasgn', '1.4', '2008/02/13', '12:57:47', 'mauro', 'classes/
@xydata/', '8ebd0d2785e732098f07cafa1275f8fb'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/
subsref.m', 'subsref', '1.7', '2008/02/13', '12:57:47', 'mauro', 'classes/@xydata/',
'4a7bf9e8296c0419568cc1473c96ae65'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xydata/
xydata.m', 'xydata', '1.14', '2008/02/13', '12:58:11', 'mauro', 'classes/@',
'ad5b52e8c6e673a5c8a7ec7e14c75f66'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xyzdata/
display.m', 'display', '1.3', '2008/02/13', '13:22:45', 'mauro', 'classes/
@xyzdata/', 'a5022342db15d74344225bd49035d90a'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xyzdata/
eq.m', 'eq', '1.3', '2008/02/18', '14:41:14', 'ingo', 'classes/@xyzdata/',
'5d594605401a152fa663fb36b4593e5b'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xyzdata/
get.m', 'get', '1.2', '2008/02/11', '17:28:46', 'ingo', 'classes/@xyzdata/',
'2272882f50ebe20a14ed82860ac3d9bc'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xyzdata/
get_xyz_values.m', 'get_xyz_values', '1.2', '2008/02/13', '13:22:45', 'mauro',
```

```
'classes/@xyzdata/', 'c2db552db0ac9e432171d720d49aec54'; ...
0417         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xyzdata/
isfield.m', 'isfield', '1.3', '2008/02/15', '16:27:16', 'mauro', 'classes/
@xyzdata/', 'f5d0e58a4c2da6506e65886ceb606cff'; ...
0418         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xyzdata/
isvalid.m', 'isvalid', '1.1', '2008/02/14', '13:48:34', 'ingo', 'classes/@xyzdata/',
'487516a270a7492c793eb68495b23c49'; ...
0419         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xyzdata/
ne.m', 'ne', '1.3', '2008/02/18', '15:12:58', 'ingo', 'classes/@xyzdata/',
'8d6ceddc17eb629d9b690f223ae29348'; ...
0420         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xyzdata/
save.m', 'save', '1.4', '2008/02/13', '13:46:45', 'mauro', 'classes/@xyzdata/',
'efe1fa514c744f87038113d8f55eb148'; ...
0421         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xyzdata/
set.m', 'set', '1.2', '2008/02/12', '15:20:33', 'ingo', 'classes/@xyzdata/',
'ddf6e478aa1257ef6bd63960f8730d8b'; ...
0422         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xyzdata/
single_operation.m', 'single_operation', '1.2', '2008/02/13', '13:22:45', 'mauro',
'classes/@xyzdata/', '5ca71a4e59027378af5bda64cf287734'; ...
0423         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xyzdata/
subsasgn.m', 'subsasgn', '1.2', '2008/02/13', '13:22:45', 'mauro', 'classes/
@xyzdata/', '1533877aa1d6eee9bdca3d9bd8c7db99'; ...
0424         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xyzdata/
subsref.m', 'subsref', '1.2', '2008/02/13', '13:22:45', 'mauro', 'classes/
@xyzdata/', 'f5ec3725f0d1a15d0cc56b4a0dd26e49'; ...
0425         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@xyzdata/
xyzdata.m', 'xyzdata', '1.3', '2008/02/13', '13:22:45', 'mauro', 'classes/@',
'14ce7be75ab04bc41ed125b11ff88d5e'; ...
0426         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/cat.
m', 'cat', '1.4', '2008/02/12', '19:54:16', 'hewitson', 'classes/@zero/',
'08227440e587634c211137c82ca028cc'; ...
0427         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/char.
m', 'char', '1.4', '2008/02/12', '19:54:16', 'hewitson', 'classes/@zero/',
'1a144dc3163aa3b62cf818323492ccb3'; ...
0428         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/
cz2iir.m', 'cz2iir', '1.5', '2008/02/12', '19:54:16', 'hewitson', 'classes/@zero/',
'19f06b6aa8a22e0e69d1e7ab5729c672'; ...
0429         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/
display.m', 'display', '1.5', '2008/02/12', '19:54:16', 'hewitson', 'classes/
@zero/', '8bafc63e51d6bf839813a0d2014e39c4'; ...
0430         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/eq.
m', 'eq', '1.5', '2008/02/18', '14:41:14', 'ingo', 'classes/@zero/',
'e2f54e0079cf64c7ddaf417341ad3511'; ...
0431         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/get.
m', 'get', '1.5', '2008/02/11', '17:28:46', 'ingo', 'classes/@zero/',
'efe8dceaae31b61e3c41d5d07b44a857'; ...
0432         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/
isfield.m', 'isfield', '1.4', '2008/02/15', '16:27:16', 'mauro', 'classes/@zero/',
'8ba4a98542d8e237f6c1315c053e8a21'; ...
0433         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/
isvalid.m', 'isvalid', '1.1', '2008/02/14', '13:48:34', 'ingo', 'classes/@zero/',
'f8cdc23a55a67de133bb09990ac69ea1'; ...
0434         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/ne.
```

```
m', 'ne', '1.4', '2008/02/18', '15:12:58', 'ingo', 'classes/@zero/',
'72320feb6fc881d25511202a38951554'; ...
0435        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/
private/pfq2ri.m', 'pfq2ri', '1.9', '2008/02/13', '10:13:54', 'ingo', 'classes/@zero/
private/', 'fd3823c71597fbb2d52a817ea1732548'; ...
0436        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/
private/pri2fq.m', 'pri2fq', '1.3', '2008/01/17', '15:59:42', 'anneke', 'classes/
@zero/private/', '174bc3076a77720840d95a39bbaff75b'; ...
0437        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/
private/retrieve.m', 'retrieve', '1.1', '2007/08/31', '17:40:08', 'hewitson',
'classes/@zero/private/', '09b762ea87be2420da51ad8a30921af4'; ...
0438        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/resp.
m', 'resp', '1.7', '2008/02/14', '14:31:38', 'mauro', 'classes/@zero/',
'e739594b6116fe7ec97b018c2f3cdbbd'; ...
0439        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/
rz2iir.m', 'rz2iir', '1.6', '2008/02/12', '19:54:16', 'hewitson', 'classes/@zero/',
'aa7d6801916d2f3e7b6686e77fa97563'; ...
0440        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/save.
m', 'save', '1.4', '2008/02/15', '16:27:16', 'mauro', 'classes/@zero/',
'd41fbc40bef41520cfef397b0d6df89f'; ...
0441        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/set.
m', 'set', '1.10', '2008/02/16', '22:22:42', 'hewitson', 'classes/@zero/',
'052e23d48389678e23ee1657dc82ede9'; ...
0442        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/
string.m', 'string', '1.7', '2008/02/15', '17:38:59', 'ingo', 'classes/@zero/',
'42cc4a675f698beba1d214faeaf281a2'; ...
0443        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/
subsasgn.m', 'subsasgn', '1.2', '2008/02/12', '19:54:16', 'hewitson', 'classes/
@zero/', '47014c0e5372a660cd0fa95a1cd5b709'; ...
0444        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/
subsref.m', 'subsref', '1.3', '2008/02/12', '19:54:16', 'hewitson', 'classes/
@zero/', '565cfcaa837db9553c1132b416e355cc'; ...
0445        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//classes/@zero/zero.
m', 'zero', '1.20', '2008/02/17', '11:38:51', 'hewitson', 'classes/@',
'97b0f7189257fbe643c7d4e94640405f'; ...
0446        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
ao_class_test.m', 'ao_class_test', '1.5', '2008/01/22', '20:45:22', 'hewitson',
'examples/', '4b8c5b690f98ab89a851ef3f6a5173d6'; ...
0447        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/example_1.
m', 'example_1', '1.18', '2008/01/21', '11:41:15', 'hewitson', 'examples/',
'ac46d16853587a166d8cc16e8378f6fe'; ...
0448        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/example_2.
m', 'example_2', '1.22', '2008/01/21', '11:41:15', 'hewitson', 'examples/',
'f1bba826ee2c3da4173f5412ff2b82ff'; ...
0449        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
fsdata_class_test.m', 'fsdata_class_test', '1.5', '2007/05/23', '07:09:31',
'hewitson', 'examples/', 'a42366d285aa847635de734152cc7dcb'; ...
0450        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
history_class_test.m', 'history_class_test', '1.5', '2008/01/22', '20:45:22',
'hewitson', 'examples/', '3dc2152cda417d417d56697e6e0e5da6'; ...
0451        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
history_class_test_2.m', 'history_class_test_2', '1.3', '2007/04/27', '06:43:06',
'hewitson', 'examples/', 'fb101fc1f98bb57cb7a14dea3554f390'; ...
```

```
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
make_test_ascii_file.m', 'make_test_ascii_file', '1.3', '2007/02/11', '19:51:11',
'hewitson', 'examples/', '8fa79484bd1614474a455ff7fedd2c63'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/run_tests.
m', 'run_tests', '1.23', '2007/07/26', '13:04:42', 'ingo', 'examples/',
'caf6d321ab142e68ffbdad8995d57969'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_abs.
m', 'test_abs', '1.7', '2008/01/21', '11:40:57', 'hewitson', 'examples/',
'73f6541812171a7476bd976eabd65410'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_acos.
m', 'test_acos', '1.2', '2007/10/08', '12:38:29', 'ingo', 'examples/',
'47a9a8af837378c335d22e7f7f98707b'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_ao_1.
m', 'test_ao_1', '1.20', '2008/01/21', '11:40:57', 'hewitson', 'examples/',
'2eb21300bb7f9b9e03a38deb2dbdf000'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ao_decimate.m', 'test_ao_decimate', '1.10', '2008/02/08', '15:00:34', 'anneke',
'examples/', '7cb61c7c7901ef80c642e0152e6ecce5'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ao_find.m', 'test_ao_find', '1.7', '2008/01/21', '11:40:57', 'hewitson',
'examples/', '311266f98140c95ba50fb433d1d6c00d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ao_hist.m', 'test_ao_hist', '1.1', '2007/05/24', '12:42:39', 'hewitson',
'examples/', '4a930d5ee305606b3ec6c544a80b2a13'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ao_interp.m', 'test_ao_interp', '1.8', '2008/01/21', '11:40:57', 'hewitson',
'examples/', 'eeda301962fa2609a75c531b9de3506a'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ao_join_ts.m', 'test_ao_join_ts', '1.5', '2008/01/21', '11:40:57', 'hewitson',
'examples/', '5d4364fd8f8c8b16fc819aa345f6f02f'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ao_polyfit.m', 'test_ao_polyfit', '1.8', '2008/01/21', '11:40:57', 'hewitson',
'examples/', '71fde483427116099afab675b2f2d7d4'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ao_select.m', 'test_ao_select', '1.6', '2008/01/21', '11:40:57', 'hewitson',
'examples/', '7919b9bcc47e823eb67d91dfc2b49da6'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ao_split.m', 'test_ao_split', '1.12', '2008/01/21', '11:40:57', 'hewitson',
'examples/', 'ca21f3711a3aeed23ad59a570186c555'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ao_split_frequency.m', 'test_ao_split_frequency', '1.9', '2008/01/21',
'11:40:57', 'hewitson', 'examples/', 'faf18f3b24ee2f8cfd28bcac00897d90'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ao_tsfcn.m', 'test_ao_tsfcn', '1.6', '2008/01/21', '11:40:57', 'hewitson',
'examples/', '1c9f4800970f73f23e73d37faf3cbd9f'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ao_waveform.m', 'test_ao_waveform', '1.4', '2008/01/21', '11:40:57',
'hewitson', 'examples/', '9049058a3f8305a52d28cd41628f15e2'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_asin.
m', 'test_asin', '1.2', '2007/10/08', '12:38:29', 'ingo', 'examples/',
'4175c246a7267624b62c6357faa56f0c'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_atan.
m', 'test_atan', '1.2', '2007/10/08', '12:38:29', 'ingo', 'examples/',
```

```
'703a7cf53723b78ca1a087825f7e4027'; ...
0470         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_cdata_class.m', 'test_cdata_class', '1.7', '2008/01/21', '11:40:57',
'hewitson', 'examples/', '0c08369f774b8de146a8d40d3c55436e'; ...
0471         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_conj.
m', 'test_conj', '1.7', '2008/01/21', '11:40:57', 'hewitson', 'examples/',
'6c96a421d3dd485a8e1c13dac2f61d56'; ...
0472         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_cos.
m', 'test_cos', '1.2', '2007/10/08', '12:38:29', 'ingo', 'examples/',
'c28a83eede59efd276a38f5d073aef46'; ...
0473         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ctranspose.m', 'test_ctranspose', '1.6', '2008/01/21', '11:40:56', 'hewitson',
'examples/', '3cc3bc6ac408ae5bc0d64585680a07d2'; ...
0474         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_det.
m', 'test_det', '1.6', '2008/01/21', '11:40:57', 'hewitson', 'examples/',
'cec7205108c7b366736dea28321d3f90'; ...
0475         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_diag.
m', 'test_diag', '1.6', '2008/01/21', '11:40:57', 'hewitson', 'examples/',
'861f86fd3a935c3e78b60cc62feef4b1'; ...
0476         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_eig.
m', 'test_eig', '1.6', '2008/01/21', '11:40:57', 'hewitson', 'examples/',
'd271302d75f852c1d433c4a020cbb2e6'; ...
0477         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_exp.
m', 'test_exp', '1.2', '2007/10/12', '12:33:08', 'ingo', 'examples/',
'83271aea38dafb46853dce15ad76971b'; ...
0478         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_fft.
m', 'test_fft', '1.7', '2008/01/21', '11:40:57', 'hewitson', 'examples/',
'613fb1d5dd2a66a47f8260022e1ae1ab'; ...
0479         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_filter_edges.m', 'test_filter_edges', '1.3', '2008/01/21', '11:40:57',
'hewitson', 'examples/', '848b40f262855358cf790e0788781094'; ...
0480         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_fir_filter.m', 'test_fir_filter', '1.7', '2008/01/21', '11:40:57', 'hewitson',
'examples/', '622a80d8822380191736184528baa41d'; ...
0481         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_iir_filtering.m', 'test_iir_filtering', '1.9', '2008/01/22', '20:45:22',
'hewitson', 'examples/', '4628818e577df9007cc76b8b3416a497'; ...
0482         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_inv.
m', 'test_inv', '1.6', '2008/01/21', '11:40:57', 'hewitson', 'examples/',
'c9afccb522ab444cb19dac4f3fe0caf2'; ...
0483         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_lincom_cdata.m', 'test_lincom_cdata', '1.10', '2008/01/21', '11:40:56',
'hewitson', 'examples/', '16033e78a89d4962bd4b2ce0824ba5be'; ...
0484         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_list.
m', 'test_list', '1.17', '2008/01/28', '10:52:20', 'josep', 'examples/',
'40edff7905e6640e2297a088f17e2aec'; ...
0485         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_log10.m', 'test_log10', '1.2', '2007/10/12', '12:33:08', 'ingo', 'examples/',
'24b6aa5cee30799984b92e3b7dd36aed'; ...
0486         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_log_ln.m', 'test_log_ln', '1.2', '2007/10/12', '12:33:08', 'ingo', 'examples/',
'6d91df2000d802236745ad1a01345e61'; ...
0487         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_lpsd.
```

```
m', 'test_lpsd', '1.18', '2008/01/21', '11:40:57', 'hewitson', 'examples/',
'e43404519016a16e15c3104b9f652312'; ...
0488        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ltpda_cohere.m', 'test_ltpda_cohere', '1.15', '2008/01/21', '11:40:57',
'hewitson', 'examples/', 'ddb8f49d88e2a2829154659a3da321e3'; ...
0489        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ltpda_cpsd.m', 'test_ltpda_cpsd', '1.14', '2008/01/21', '11:40:57', 'hewitson',
'examples/', '0ced1cc51a5a28c0ffa37a0dab38be7b'; ...
0490        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ltpda_gapfilling.m', 'test_ltpda_gapfilling', '1.1', '2008/01/28', '10:43:50',
'josep', 'examples/', '138d7faa1d40754a89a225f9da28257d'; ...
0491        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ltpda_lincom.m', 'test_ltpda_lincom', '1.13', '2008/01/21', '11:40:57',
'hewitson', 'examples/', 'f9864f356e2f63be3f69e9327be549a4'; ...
0492        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ltpda_linedetect.m', 'test_ltpda_linedetect', '1.10', '2008/01/21', '11:40:57',
'hewitson', 'examples/', '4a054af3a4f244872d287dc39bc5ecda'; ...
0493        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ltpda_ltfe.m', 'test_ltpda_ltfe', '1.13', '2008/02/20', '07:25:48', 'hewitson',
'examples/', 'aff033df1aec7ce60280426922b6490e'; ...
0494        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ltpda_nfest.m', 'test_ltpda_nfest', '1.9', '2008/01/21', '11:40:57',
'hewitson', 'examples/', '145b0f29830e188c9f465d8e48208c4d'; ...
0495        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ltpda_polydetrend.m', 'test_ltpda_polydetrend', '1.13', '2008/01/21',
'11:40:57', 'hewitson', 'examples/', '6b62abf9bf7db9a01e2ac8e32408d4a6'; ...
0496        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ltpda_pwelch.m', 'test_ltpda_pwelch', '1.21', '2008/02/20', '07:39:01',
'hewitson', 'examples/', 'e4e7ad564cae65a05295fb73825d3b42'; ...
0497        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ltpda_spikecleaning.m', 'test_ltpda_spikecleaning', '1.1', '2008/01/26',
'16:49:24', 'josep', 'examples/', '7397a6e0e5a666bbb1b1277663d28a53'; ...
0498        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ltpda_tfe.m', 'test_ltpda_tfe', '1.17', '2008/01/21', '11:40:57', 'hewitson',
'examples/', 'f0ff8e16a1473d27fdb1cc00de32adf3'; ...
0499        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_ltpda_xcorr.m', 'test_ltpda_xcorr', '1.6', '2008/01/21', '11:40:57',
'hewitson', 'examples/', 'a590bf098beedf48f3333f7109ca526a'; ...
0500        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_mean.
m', 'test_mean', '1.8', '2008/01/21', '11:40:57', 'hewitson', 'examples/',
'a2fabb43f98842cf9d6437c51760332a'; ...
0501        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_median.m', 'test_median', '1.2', '2007/10/12', '12:33:08', 'ingo', 'examples/',
'697761c15cc1a5e5e4358e9e9fa897a0'; ...
0502        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_mfir_class.m', 'test_mfir_class', '1.3', '2008/02/18', '18:08:51', 'mauro',
'examples/', '14c1555ec8138e534aad25ac9b2f3d8b'; ...
0503        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_miir_class.m', 'test_miir_class', '1.14', '2008/02/18', '18:08:51', 'mauro',
'examples/', 'e3256dda1d1fb845294bae2b7ba38481'; ...
0504        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_miir_filter.m', 'test_miir_filter', '1.18', '2008/01/21', '11:40:57',
'hewitson', 'examples/', 'ef37f1e3e362c309cea9e0ea1514fc48'; ...
```

```
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_miir_filtfilt.m', 'test_miir_filtfilt', '1.13', '2008/01/21', '11:40:56',
'hewitson', 'examples/', '23f0ae05ad17e2c405fa32db9a8629c3'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_miir_redesign.m', 'test_miir_redesign', '1.6', '2008/01/21', '11:40:56',
'hewitson', 'examples/', '7486c7649efe330ee2453bd422e3d0ed'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_minus.m', 'test_minus', '1.9', '2008/01/21', '11:40:57', 'hewitson',
'examples/', 'c7132f53ff1f8509e07d764bb8866027'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_mpower.m', 'test_mpower', '1.9', '2008/01/21', '11:40:57', 'hewitson',
'examples/', '52725fac733b009b04f0a8ca50779d09'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_noisegen.m', 'test_noisegen', '1.9', '2008/01/21', '11:40:57', 'hewitson',
'examples/', '2c6d9da5f820da7a7e7b9780a992c7b0'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_norm.
m', 'test_norm', '1.8', '2008/01/21', '11:40:56', 'hewitson', 'examples/',
'e0bf4f34167cb77bb7eb636cc3b0b61b'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_plist_string.m', 'test_plist_string', '1.2', '2007/04/27', '06:43:06',
'hewitson', 'examples/', 'b31df094b722a629eaaa480286d2f3ba'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_plus.
m', 'test_plus', '1.10', '2008/01/21', '11:40:57', 'hewitson', 'examples/',
'e71776a29130a74e193316768087f224'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_pzm_to_fir.m', 'test_pzm_to_fir', '1.1', '2007/08/16', '14:59:34', 'hewitson',
'examples/', '62ddb026ec03d246b3a17ad8c178bccd'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_pzmodel_class.m', 'test_pzmodel_class', '1.12', '2008/01/21', '11:40:57',
'hewitson', 'examples/', '27c3f41379e8b2f9d31f1ee350f71b3d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_rdivide.m', 'test_rdivide', '1.8', '2008/01/21', '11:40:57', 'hewitson',
'examples/', '04d879e7bea0faba343a880643bc2060'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_recreate_1.m', 'test_recreate_1', '1.12', '2008/01/22', '20:45:22', 'hewitson',
'examples/', '822b201b35ddff19c4f70534dd165a41'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_resample.m', 'test_resample', '1.6', '2008/01/21', '11:40:57', 'hewitson',
'examples/', '02aa07c5e250a21deaf0a6ea06891639'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_simulated_data.m', 'test_simulated_data', '1.8', '2008/01/21', '11:40:57',
'hewitson', 'examples/', 'e3470058e12f39feb3845b64d5b98a4c'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_sin.
m', 'test_sin', '1.2', '2007/10/08', '12:38:29', 'ingo', 'examples/',
'20151ef3baad9a3b12a6edfeeb852848'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_specwin_class.m', 'test_specwin_class', '1.4', '2008/01/21', '11:40:57',
'hewitson', 'examples/', '8524e77dbb1e55dcf1543c5b4434fd96'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_sqrt.
m', 'test_sqrt', '1.3', '2007/10/12', '12:33:08', 'ingo', 'examples/',
'962420f6c7e004109ff0fe7f42508617'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_std.
m', 'test_std', '1.2', '2007/10/12', '12:33:08', 'ingo', 'examples/',
```

```
'd3a3e1fa85f7106ee4e64c5bbd943f00'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_svd.
m', 'test_svd', '1.7', '2008/01/21', '11:40:57', 'hewitson', 'examples/',
'dceed7d62fe7953d9e983519359ef54b'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_tan.
m', 'test_tan', '1.2', '2007/10/12', '12:33:08', 'ingo', 'examples/',
'903812c6b95cb9ea009b4d108a9bb4d6'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_times.m', 'test_times', '1.10', '2008/01/21', '11:40:57', 'hewitson',
'examples/', '7827e7715a2ff002a2e692bca4fc6698'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_transpose.m', 'test_transpose', '1.6', '2008/01/21', '11:40:57', 'hewitson',
'examples/', '5655d4f6551a457b4b08fb7b2384032d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/test_var.
m', 'test_var', '1.2', '2007/10/12', '12:33:08', 'ingo', 'examples/',
'dc08377a76f550a6616e0f75c7a25169'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_xml_complex.m', 'test_xml_complex', '1.8', '2008/01/21', '11:40:57',
'hewitson', 'examples/', '5546ea84f721a5a49e44759d1fbaaa57'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
test_xydata_class.m', 'test_xydata_class', '1.7', '2008/01/21', '11:40:57',
'hewitson', 'examples/', 'a51f77df03b8a41a5e9303679a791542'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
testing_xml.m', 'testing_xml', '1.18', '2008/01/22', '20:45:22', 'hewitson',
'examples/', 'd60c9b1b94423cb7327dc926a0345608'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//examples/
tsdata_class_test.m', 'tsdata_class_test', '1.3', '2007/04/27', '06:43:06',
'hewitson', 'examples/', '2fc3ec4d5eabce66b112226f9d5d7d14'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//help/convertTOC.
m', 'convertTOC', '1.1', '2007/07/24', '08:15:06', 'hewitson', 'help/',
'4e89c260b407812e617de3baba8a0a5f'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//help/read_item.m',
'read_item', '1.2', '2007/08/14', '09:23:01', 'ingo', 'help/',
'817625a975768b098eb4b38897311007'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//help/runcmd.m',
'runcmd', '1.2', '2007/10/24', '10:59:50', 'ingo', 'help/',
'ecfc4984b1d6192f18aad65e8b0e1b54'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//help/ug/
mk_specwin_table.m', 'mk_specwin_table', '1.2', '2008/01/22', '20:45:22',
'hewitson', 'help/ug/', '3ee0686c3a6dd85bc68e94dc5929eab1'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/analyses/
stuttgart/dkalman_matrices.m', 'dkalman_matrices', '1.2', '2008/01/22', '20:45:22',
'hewitson', 'm/analyses/stuttgart/', '7dfe7ed79ccf3a681726c8fdf2e36f89'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/analyses/
stuttgart/e2e2ao.m', 'e2e2ao', '1.2', '2008/01/22', '20:45:22', 'hewitson', 'm/
analyses/stuttgart/', '21ce5d1be0d906206098ba875dcb54d5'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/analyses/
stuttgart/ltpda_diff_acc.m', 'ltpda_diff_acc', '1.3', '2007/11/07', '15:57:27',
'ingo', 'm/analyses/stuttgart/', '4f55adaa4e4a6c8f8ac094a3e0fcde09'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/analyses/
stuttgart/ltpda_dkalman.m', 'ltpda_dkalman', '1.4', '2007/11/07', '15:57:27',
'ingo', 'm/analyses/stuttgart/', '96c323c76a034e646203dab4af0d8902'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/analyses/
```

```
stuttgart/test_dkalman.m', 'test_dkalman', '1.4', '2007/11/07', '15:57:27', 'ingo',
'm/analyses/stuttgart/', '1f72515a5284bfe05110b5dbd45de9f2'; ...
0541        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/etc/
ao_fcn_template.m', 'ao_fcn_template', '1.7', '2008/01/15', '06:24:51', 'mauro', 'm/
etc/', 'cde9fc596b148f07e5d9d85bde2c1131'; ...
0542        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/etc/
ltpda_startup.m', 'ltpda_startup', '1.22', '2008/02/13', '14:39:34', 'mauro', 'm/
etc/', '53118e02a7822b0b949362152765ea30'; ...
0543        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/etc/
ltpda_user_fcn_template.m', 'ltpda_user_fcn_template', '1.3', '2008/02/14',
'14:32:16', 'mauro', 'm/etc/', '940f7b54760296c363ae56194e963fbd'; ...
0544        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/etc/
method_fcn_template.m', 'method_fcn_template', '1.3', '2008/02/15', '16:25:38',
'mauro', 'm/etc/', 'afd3eba4db9d8a12264a98d1e517166e'; ...
0545        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ao_browser/
explore_ao.m', 'explore_ao', '1.14', '2008/02/14', '10:41:02', 'ingo', 'm/gui/
ao_browser/', '828d083cc6a659f17d5d7600864df365'; ...
0546        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ao_browser/
ltpda_javacomponent.m', 'ltpda_javacomponent', '1.2', '2008/01/22', '20:45:22',
'hewitson', 'm/gui/ao_browser/', 'b6acee252c6e53887f4e1b7e0708847d'; ...
0547        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ao_browser/
ltpda_uitree.m', 'ltpda_uitree', '1.2', '2008/01/22', '20:45:22', 'hewitson', 'm/gui/
ao_browser/', '863ff08e050a8562f1dcc59151800aa5'; ...
0548        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ao_browser/
ltpda_uitreenode.m', 'ltpda_uitreenode', '1.2', '2008/01/22', '20:45:22',
'hewitson', 'm/gui/ao_browser/', '77f4c333c831ba4abdf9d51383e3d0d4'; ...
0549        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/buildConditions.m', 'buildConditions', '1.3', '2008/01/22', '20:45:22',
'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/', '9a703c0a129af59efface18fa11c342f'; ...
0550        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/buildquery.m', 'buildquery', '1.2', '2008/01/22', '20:45:22', 'hewitson',
'm/gui/ltpdaRepoGUI/callbacks/', '41836dde0497e7e0d614b299310466d7'; ...
0551        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_classMethodList.m', 'cb_classMethodList', '1.1', '2008/01/22',
'21:10:23', 'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/',
'5ec7b8e46b4a250e299f9f6594c9ef18'; ...
0552        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_condPlusBtn.m', 'cb_condPlusBtn', '1.2', '2008/01/22', '20:45:22',
'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/', '76b039c38474efeb440d0945cdef9449'; ...
0553        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_condSubtractBtn.m', 'cb_condSubtractBtn', '1.2', '2008/01/22',
'20:45:22', 'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/',
'ddb42ba8a6740fc0c88604369a4794cc'; ...
0554        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_connectBtn.m', 'cb_connectBtn', '1.2', '2008/01/22', '20:45:22',
'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/', '6d07f4985798cc962aeb4d38e944339f'; ...
0555        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_dbSelect.m', 'cb_dbSelect', '1.2', '2008/01/22', '20:45:22',
'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/', '709e3f404abef54748e212c48e290752'; ...
0556        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_executeQuery.m', 'cb_executeQuery', '1.3', '2008/01/22', '20:45:22',
'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/', '3d6a9460b7cd6ca4f81d2e1dad940f21'; ...
0557        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
```

```
callbacks/cb_fieldSelect.m', 'cb_fieldSelect', '1.3', '2008/02/07', '19:27:37',
'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/', '447eda8ac47e30cb997b97da50af0ce0'; ...
0558        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_getDBs.m', 'cb_getDBs', '1.2', '2008/01/22', '20:45:22', 'hewitson', 'm/
gui/ltpdaRepoGUI/callbacks/', '00916e5405fa39f1c4bae1cfc7e2d1e8'; ...
0559        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_mainfigClose.m', 'cb_mainfigClose', '1.2', '2008/01/22', '20:45:22',
'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/', 'f6d210abc94b3a46fb610d3712d6e6f3'; ...
0560        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_refreshObjsList.m', 'cb_refreshObjsList', '1.2', '2008/01/22',
'20:45:22', 'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/',
'24c7097cc33f392ce90447e3baf582b3'; ...
0561        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_refreshObjsListQL.m', 'cb_refreshObjsListQL', '1.1', '2008/01/22',
'21:10:23', 'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/',
'040fa0a24049aa513f7458ac61ebc39f'; ...
0562        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_retrieveBtn.m', 'cb_retrieveBtn', '1.2', '2008/01/22', '20:45:22',
'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/', '83c4d30ceb8fcd901ce1f376d71ea081'; ...
0563        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_serverSelect.m', 'cb_serverSelect', '1.2', '2008/01/22', '20:45:22',
'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/', '85f8fcb784fac30a87e81442529b5a98'; ...
0564        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_submitBtn.m', 'cb_submitBtn', '1.2', '2008/01/22', '20:45:22',
'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/', '38d908e7dc01f8209a01e02037c95566'; ...
0565        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_tableSelect.m', 'cb_tableSelect', '1.2', '2008/01/22', '20:45:22',
'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/', '0a7fe8d4af4b28f39cad320764f5a155'; ...
0566        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_workspaceObjsList.m', 'cb_workspaceObjsList', '1.2', '2008/01/22',
'20:45:22', 'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/',
'dd15b15df3159f11158aacc2a05dae94'; ...
0567        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/cb_workspaceObjsListQL.m', 'cb_workspaceObjsListQL', '1.1', '2008/01/22',
'21:10:23', 'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/',
'8ebb00a37a7e0405cb9c989f910a3f03'; ...
0568        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/fillClassMethodsList.m', 'fillClassMethodsList', '1.1', '2008/01/22',
'21:10:23', 'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/',
'9aa8fd46f9af4bf74258bdfbbb6fa254'; ...
0569        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/getFields.m', 'getFields', '1.2', '2008/01/22', '20:45:22', 'hewitson', 'm/
gui/ltpdaRepoGUI/callbacks/', 'c33b03fa9c0cf88ed4d2647deb27ffeb'; ...
0570        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/getTables.m', 'getTables', '1.2', '2008/01/22', '20:45:22', 'hewitson', 'm/
gui/ltpdaRepoGUI/callbacks/', '915fd622778f354fd4ec3395665cf2ef'; ...
0571        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/getWorkspaceObjs.m', 'getWorkspaceObjs', '', '', '', '', 'm/gui/
ltpdaRepoGUI/callbacks/', '4553e6a4711a0fad35433e665ac8d191'; ...
0572        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/setWorkspaceObjsList.m', 'setWorkspaceObjsList', '1.2', '2008/01/22',
'20:45:22', 'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/',
'b30ce3f1b60370c2dae2105837c6dfa8'; ...
```

```
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
callbacks/setWorkspaceObjsListQL.m', 'setWorkspaceObjsListQL', '1.1', '2008/01/22',
'21:10:23', 'hewitson', 'm/gui/ltpdaRepoGUI/callbacks/',
'7fb8175239526929cabe1e9b5c2a272c'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
muitable.m', 'muitable', '1.2', '2008/01/22', '20:45:22', 'hewitson', 'm/gui/
ltpdaRepoGUI/', 'c3e3c1c148124a23a9e07ea9bd67d400'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
repogui.m', 'repogui', '1.6', '2008/01/22', '20:45:22', 'hewitson', 'm/gui/
ltpdaRepoGUI/', '5beabe94e27dda9809d7fb40c139293f'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
sqlResultsGUI.m', 'sqlResultsGUI', '1.4', '2008/02/14', '09:06:57', 'hewitson', 'm/
gui/ltpdaRepoGUI/', '22311ca654d1635130ddbfe4cb06cb2d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/ltpdaRepoGUI/
uitabpanel.m', 'uitabpanel', '1.2', '2008/01/22', '20:45:22', 'hewitson', 'm/gui/
ltpdaRepoGUI/', 'bafa3bbcea7974a63bbda5b78b6dde2e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/gui/
pzmodel_designer/pzmodel_helper.m', 'pzmodel_helper', '1.8', '2008/01/22',
'20:45:23', 'hewitson', 'm/gui/pzmodel_designer/',
'9d4040abfdb5091508d85e82d59fa50d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
collect_inputs.m', 'collect_inputs', '1.1', '2008/01/29', '12:30:36', 'mauro', 'm/
helper/', '579f0e902090ae9c0d4aacf1266a320a'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/gcbsh.m',
'gcbsh', '1.3', '2008/02/14', '18:43:30', 'hewitson', 'm/helper/',
'd0a0528e4f96f7019957019bb647de7c'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
generic_class_fcn/generic_eq.m', 'generic_eq', '1.5', '2008/02/19', '10:01:41',
'ingo', 'm/helper/generic_class_fcn/', 'a5fb36565a7527a933970908d10d6364'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
generic_class_fcn/generic_get.m', 'generic_get', '1.2', '2008/02/11', '21:21:56',
'hewitson', 'm/helper/generic_class_fcn/', '18a5b5f5e542e9d311bd28fc53f3d7d5'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
generic_class_fcn/generic_set.m', 'generic_set', '1.2', '2008/02/12', '14:35:57',
'ingo', 'm/helper/generic_class_fcn/', '994d0c173f9a89031af3a6c248aea183'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
get_curr_m_file_path.m', 'get_curr_m_file_path', '1.2', '2007/06/25', '11:21:48',
'ingo', 'm/helper/', 'e2402c7165832067b86838ec4a2d7cb8'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_classes.m', 'ltpda_classes', '1.1', '2008/02/14', '16:03:52', 'mauro', 'm/
helper/', '1e2fdcf68195fb83a8c1f3ebd0fb6df4'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_deg2rad.m', 'ltpda_deg2rad', '1.5', '2007/07/13', '12:17:39', 'ingo', 'm/
helper/', '3260e0dfba99d522a4def445df2b634e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_fields2list.m', 'ltpda_fields2list', '1.5', '2007/07/13', '12:17:39', 'ingo',
'm/helper/', '755aac4bd5ad5b73c54120b136426f1d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_filescan.m', 'ltpda_filescan', '1.6', '2007/11/09', '14:07:39', 'ingo', 'm/
helper/', '7749aefc8579c165a51a8ea5f155d48e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_fq2ri.m', 'ltpda_fq2ri', '1.4', '2007/01/26', '14:03:24', 'hewitson', 'm/
helper/', '39f06aa1ce9629f1b8e43a3b2950a467'; ...
```

```
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_fq2ri2.m', 'ltpda_fq2ri2', '1.6', '2007/07/11', '14:06:39', 'anneke', 'm/
helper/', 'a489eca7243f68a3df3cfd36a480bde1'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_funchash.m', 'ltpda_funchash', '1.1', '2007/11/21', '13:59:29', 'hewitson', 'm/
helper/', 'ea72ffc343e6cf515b7b3da66a3236be'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_getmax.m', 'ltpda_getmax', '1.5', '2007/07/13', '12:17:39', 'ingo', 'm/
helper/', '8951761b5a1fd8e78f3486c9979591bc'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_hash.m', 'ltpda_hash', '1.1', '2007/09/15', '04:54:44', 'hewitson', 'm/
helper/', '360f81d5d18e7cc223560e96b4ce4e22'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_io_collect.m', 'ltpda_io_collect', '1.1', '2007/10/08', '09:35:48',
'hewitson', 'm/helper/', '1317f70f1870455c89df29d312f85f92'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_isobject.m', 'ltpda_isobject', '1.3', '2008/02/14', '16:04:30', 'mauro', 'm/
helper/', '6754de99d9b87d6c3c98cb3f0e361bbf'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_isuserobject.m', 'ltpda_isuserobject', '1.5', '2008/02/14', '16:04:30',
'mauro', 'm/helper/', 'a38bcec5781d81b22ad95359793625e9'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_label.m', 'ltpda_label', '1.2', '2007/07/13', '12:17:39', 'ingo', 'm/helper/',
'f7d69bfc076dcc4c9d3c138ab164ed64'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_mat2str.m', 'ltpda_mat2str', '1.5', '2008/02/19', '17:28:38', 'ingo', 'm/
helper/', '0ea1a1819dd58a7ebc6e28d2cc853e70'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_matchvectors.m', 'ltpda_matchvectors', '1.3', '2007/08/14', '09:25:24',
'ingo', 'm/helper/', 'fcbfffe1996678db901e9f6215592e5e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_num2str.m', 'ltpda_num2str', '1.2', '2007/10/26', '14:39:54', 'hewitson', 'm/
helper/', '9de091f63164c3d5dc20a1e6a6cf4e5c'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_phase.m', 'ltpda_phase', '1.5', '2007/07/13', '12:17:39', 'ingo', 'm/helper/',
'3db03b95601c07a31ada4afe5bb1b27e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_rand.m', 'ltpda_rand', '1.2', '2007/07/13', '12:17:39', 'ingo', 'm/helper/',
'be0afcec0e8600ad408605536bd3bc92'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_ri2fq.m', 'ltpda_ri2fq', '1.4', '2007/01/26', '14:03:24', 'hewitson', 'm/
helper/', '3de354746004c3b68bd48121d597a24d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_saveobj.m', 'ltpda_saveobj', '1.2', '2008/02/13', '10:13:54', 'ingo', 'm/
helper/', '341c28af4f2c7165776f0a53a8ab817e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_str2cells.m', 'ltpda_str2cells', '1.7', '2007/07/13', '12:17:39', 'ingo', 'm/
helper/', 'bde46aca18d8eff9440597a052724db1'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_strs2cells.m', 'ltpda_strs2cells', '1.5', '2007/07/13', '12:17:39', 'ingo', 'm/
helper/', 'd298012f0824a936f572d1187b25c148'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_structcat.m', 'ltpda_structcat', '1.5', '2007/07/13', '12:17:39', 'ingo', 'm/
```

```
helper/', '42cf86c7974ffcc2ea7f8ab73cd14619'; ...
0608        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_structidx.m', 'ltpda_structidx', '1.5', '2007/07/13', '12:17:39', 'ingo', 'm/
helper/', '0978884418620c65039f67d7fb0ce145'; ...
0609        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_unwrapdeg.m', 'ltpda_unwrapdeg', '1.5', '2007/07/13', '12:17:39', 'ingo', 'm/
helper/', '9ba9b98e623704364cbdeebc54cb312c'; ...
0610        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_user_classes.m', 'ltpda_user_classes', '1.1', '2008/02/14', '16:03:52',
'mauro', 'm/helper/', '7e0e734a2b5aa4f89886a948f43e57a2'; ...
0611        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_xml_read_datasamples.m', 'ltpda_xml_read_datasamples', '1.5', '2007/10/26',
'14:39:54', 'hewitson', 'm/helper/', '823691d137e7db62f7a6aec7b1d81049'; ...
0612        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_xmlread.m', 'ltpda_xmlread', '1.5', '2008/02/18', '13:37:53', 'ingo', 'm/
helper/', '6297603f469ea60bedd4f47e35ed99f8'; ...
0613        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
ltpda_xmlwrite.m', 'ltpda_xmlwrite', '1.3', '2008/02/15', '13:29:17', 'ingo', 'm/
helper/', 'ce0ef37ac01b9ed5d19e8801ff64de04'; ...
0614        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/phase.m',
'phase', '1.1', '2008/01/20', '12:48:50', 'hewitson', 'm/helper/',
'5647f46324ee2aa57650f65acee1a637'; ...
0615        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/rstruct.
m', 'rstruct', '1.1', '2007/09/18', '17:44:36', 'hewitson', 'm/helper/',
'e2e5fbd9839652bbc3c0cca44bcd630c'; ...
0616        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/strpad.
m', 'strpad', '1.2', '2007/07/13', '12:17:39', 'ingo', 'm/helper/',
'9cbd84ffb8deaa6b4584519e53582bb6'; ...
0617        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/helper/
wrapstring.m', 'wrapstring', '1.4', '2007/07/13', '12:17:39', 'ingo', 'm/helper/',
'0edae957eb276ef13bc6b1c45aa4acbd'; ...
0618        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/math/
ltpda_lincom.m', 'ltpda_lincom', '1.13', '2007/11/09', '14:07:39', 'ingo', 'm/
math/', '1d5db226ff932c757523517a4118cd8a'; ...
0619        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/mysql/logindlg.
m', 'logindlg', '1.3', '2008/01/22', '20:45:22', 'hewitson', 'm/mysql/',
'a88e4de3a1b6319f78ede872b8d24adf'; ...
0620        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/mysql/
ltpda_dbquery.m', 'ltpda_dbquery', '1.4', '2007/11/13', '13:53:47', 'hewitson', 'm/
mysql/', '52ea36a691c40914e036d3ae24c757fb'; ...
0621        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/mysql/
ltpda_insert.m', 'ltpda_insert', '1.1', '2007/08/31', '17:29:22', 'hewitson', 'm/
mysql/', '3e4bd4431e54e7c33489ecbd99c01b02'; ...
0622        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/mysql/
ltpda_obj_retrieve.m', 'ltpda_obj_retrieve', '1.10', '2008/02/13', '15:13:45',
'hewitson', 'm/mysql/', '6010036ef803557e7841d813e3faee0e'; ...
0623        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/mysql/
ltpda_obj_submit.m', 'ltpda_obj_submit', '1.18', '2008/02/13', '15:13:45',
'hewitson', 'm/mysql/', 'c74b6f5a611f6276e889d166d5ca1e0e'; ...
0624        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/mysql/
mysql_connect.m', 'mysql_connect', '1.8', '2008/01/30', '21:10:58', 'hewitson', 'm/
mysql/', 'd730d420029ad27152c84f09a2d0e2be'; ...
0625        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/mysql/
```

```
mysql_getMD5hash.m', 'mysql_getMD5hash', '1.1', '2007/09/15', '04:54:44',
'hewitson', 'm/mysql/', 'c53a09cbf49a8de7be2f5c76f8d41fb2'; ...
0626         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/mysql/
mysql_getMaxId.m', 'mysql_getMaxId', '1.3', '2007/08/31', '17:29:22', 'hewitson', 'm/
mysql/', '8cf0ad5bcef61a66cdef18ff5e280cf4'; ...
0627         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/mysql/
mysql_getObjIds.m', 'mysql_getObjIds', '1.2', '2007/11/13', '09:44:40', 'hewitson',
'm/mysql/', '06312af931bb240573db52638715d5c8'; ...
0628         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/mysql/
mysql_getObjType.m', 'mysql_getObjType', '1.2', '2007/10/17', '15:01:45',
'hewitson', 'm/mysql/', 'e708c97f4961623faf6095533ebbaf45'; ...
0629         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/mysql/
mysql_getUserID.m', 'mysql_getUserID', '1.5', '2007/10/17', '15:05:23', 'hewitson',
'm/mysql/', '94350fdbff55b054b3dcc5ead903ae23'; ...
0630         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/mysql/
mysql_getXdoc.m', 'mysql_getXdoc', '1.2', '2007/10/17', '15:04:27', 'hewitson', 'm/
mysql/', '159413b31a108d666f5005ee8d230e2c'; ...
0631         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/noisegenerator/
ltpda_noisegen.m', 'ltpda_noisegen', '1.10', '2007/11/09', '14:07:39', 'ingo', 'm/
noisegenerator/', '61d0a87da3b9bc0b0c7535e99e58d151'; ...
0632         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/noisegenerator/
ngconv.m', 'ngconv', '1.2', '2008/01/23', '17:23:56', 'anneke', 'm/noisegenerator/',
'37c92b94595c3bce91273c6e62a28350'; ...
0633         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/noisegenerator/
nginit.m', 'nginit', '1.2', '2008/01/23', '17:23:56', 'anneke', 'm/noisegenerator/',
'b772b791e07a95345ba95a882b2636a3'; ...
0634         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/noisegenerator/
ngprop.m', 'ngprop', '1.4', '2008/01/23', '17:23:56', 'anneke', 'm/noisegenerator/',
'8ea9ccf697281ea72e220dc0cedc15c7'; ...
0635         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/noisegenerator/
ngsetup.m', 'ngsetup', '1.2', '2008/01/23', '17:23:56', 'anneke', 'm/
noisegenerator/', '8b25cab8782d17226a04cfa8fae00437'; ...
0636         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/noisegenerator/
private/conv_noisegen.m', 'conv_noisegen', '1.2', '2008/01/23', '17:23:56',
'anneke', 'm/noisegenerator/private/', '0907531c0edfdbc1aa14d7476dd87ad1'; ...
0637         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/noisegenerator/
private/fq2fac.m', 'fq2fac', '1.2', '2008/01/23', '17:23:56', 'anneke', 'm/
noisegenerator/private/', '1db11b8ec88971a8de3711cd71f96a00'; ...
0638         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
islinespec.m', 'islinespec', '1.3', '2007/08/24', '13:05:09', 'ingo', 'm/
plottools/', '959c0eb7239aad6175b9a74626a52678'; ...
0639         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_allgrid.m', 'ltpda_allgrid', '1.4', '2007/07/16', '12:52:20', 'ingo', 'm/
plottools/', 'e16f597fcb3cf46cdccd6e3a57248261'; ...
0640         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_allxaxis.m', 'ltpda_allxaxis', '1.6', '2007/07/30', '16:04:09', 'ingo', 'm/
plottools/', '81676149165d46066b3efc61a627b225'; ...
0641         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_allxlabel.m', 'ltpda_allxlabel', '1.5', '2007/07/30', '16:04:09', 'ingo', 'm/
plottools/', 'd2477295f265fdb4f6a933e35f8ba35b'; ...
0642         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_allxscale.m', 'ltpda_allxscale', '1.5', '2007/07/30', '16:04:09', 'ingo', 'm/
plottools/', '80c4bfd21cb168f83a5e2e1768faa025'; ...
```

```
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_allyaxis.m', 'ltpda_allyaxis', '1.5', '2007/07/30', '16:04:09', 'ingo', 'm/
plottools/', 'd4dc4d02886b4cd4c35276c372c5fe58'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_allylabel.m', 'ltpda_allylabel', '1.5', '2007/07/30', '16:04:09', 'ingo', 'm/
plottools/', '77bf4ef18ad193df1b39042c56a3734c'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_allyscale.m', 'ltpda_allyscale', '1.5', '2007/07/30', '16:04:09', 'ingo', 'm/
plottools/', '94a7e9600bbd1e2ffb75a1a5d534b94e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_cscale.m', 'ltpda_cscale', '1.4', '2007/07/16', '12:52:20', 'ingo', 'm/
plottools/', '14b7ef8e2217d2ce0c53725a8cc2aba3'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_legendAdd.m', 'ltpda_legendAdd', '1.4', '2007/06/07', '08:56:24', 'ingo', 'm/
plottools/', 'afade18d2d703a365d2040fa09591a5a'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_msuptitle.m', 'ltpda_msuptitle', '1.4', '2007/07/16', '12:52:20', 'ingo', 'm/
plottools/', '111c24f1715ad62f51af8b30d9dbaaa8'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_xaxis.m', 'ltpda_xaxis', '1.4', '2007/07/16', '12:52:20', 'ingo', 'm/
plottools/', 'dbe54375a7dd32f20c0462dcc7a559d1'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_xscale.m', 'ltpda_xscale', '1.2', '2007/07/16', '12:52:20', 'ingo', 'm/
plottools/', '5991478a4da87ff71f615ed6f63f2fc0'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_xticks.m', 'ltpda_xticks', '1.4', '2007/07/16', '12:52:20', 'ingo', 'm/
plottools/', '17978f5a401d3921adf084264d7ce5d3'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_yaxis.m', 'ltpda_yaxis', '1.4', '2007/07/16', '12:52:20', 'ingo', 'm/
plottools/', '6ae03b3feb5042c1984a0126bee5886e'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_yscale.m', 'ltpda_yscale', '1.4', '2007/07/16', '12:52:20', 'ingo', 'm/
plottools/', 'e9f4508553ecc648d4888a8ffe08170b'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_yticks.m', 'ltpda_yticks', '1.4', '2007/07/16', '12:52:20', 'ingo', 'm/
plottools/', 'e96ce89658418d9f543120e7a5c4bc6f'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_zaxis.m', 'ltpda_zaxis', '1.4', '2007/07/16', '12:52:20', 'ingo', 'm/
plottools/', '6e126b1f40a2de06abe64f5438bb0d45'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/plottools/
ltpda_zscale.m', 'ltpda_zscale', '1.4', '2007/07/16', '12:52:20', 'ingo', 'm/
plottools/', 'b769b59accf4ce89a62ef6bb88e9b630'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
frequency_domain/ltpda_cohere.m', 'ltpda_cohere', '1.10', '2007/12/18', '14:11:26',
'josep', 'm/sigproc/frequency_domain/', 'ed98e1bd00059b152c5f7fc6b3177dba'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
frequency_domain/ltpda_compute_f.m', 'ltpda_compute_f', '1.4', '2007/10/28',
'10:23:40', 'hewitson', 'm/sigproc/frequency_domain/',
'f6f359941c424a784af1c1cc6cd5fb78'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
frequency_domain/ltpda_cpsd.m', 'ltpda_cpsd', '1.7', '2007/07/16', '12:52:21',
'ingo', 'm/sigproc/frequency_domain/', 'f9fd1797707e09c7093788d4df152880'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
```

```
frequency_domain/ltpda_lcohere.m', 'ltpda_lcohere', '1.9', '2008/02/20', '09:17:28',
'hewitson', 'm/sigproc/frequency_domain/', '29a68b01d44dab98da04e31a6fc87ece'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
frequency_domain/ltpda_lcpsd.m', 'ltpda_lcpsd', '1.9', '2008/02/19', '19:27:25',
'hewitson', 'm/sigproc/frequency_domain/', '00dd0cbf168c6a7eb59be174f2fb889d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
frequency_domain/ltpda_linedetect.m', 'ltpda_linedetect', '1.8', '2007/11/08',
'10:24:54', 'hewitson', 'm/sigproc/frequency_domain/',
'3c5a39c0b737db9399d605d656244701'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
frequency_domain/ltpda_lpsd.m', 'ltpda_lpsd', '1.19', '2008/02/19', '19:19:26',
'hewitson', 'm/sigproc/frequency_domain/', '5b93fcdd4d32ffec85767bdffb104e32'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
frequency_domain/ltpda_ltf_plan.m', 'ltpda_ltf_plan', '1.1', '2008/02/19',
'09:11:54', 'hewitson', 'm/sigproc/frequency_domain/',
'253b163f6186d50bd1bac1c4f0b36fe9'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
frequency_domain/ltpda_ltfe.m', 'ltpda_ltfe', '1.14', '2008/02/20', '09:16:11',
'hewitson', 'm/sigproc/frequency_domain/', '51fee2a303d238731f8355b16bf5391f'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
frequency_domain/ltpda_lxspec.m', 'ltpda_lxspec', '1.7', '2007/10/08', '11:28:29',
'hewitson', 'm/sigproc/frequency_domain/', '8bf0d023b137d580528e3d58b530a3e0'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
frequency_domain/ltpda_nfest.m', 'ltpda_nfest', '1.7', '2007/11/08', '10:24:54',
'hewitson', 'm/sigproc/frequency_domain/', '0ac12c35d2b7f149b49bdfe9dcfe9557'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
frequency_domain/ltpda_psd.m', 'ltpda_psd', '1.4', '2008/02/20', '07:39:01',
'hewitson', 'm/sigproc/frequency_domain/', 'a745562fc842090e4b49bf7bdc38ad81'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
frequency_domain/ltpda_pwelch.m', 'ltpda_pwelch', '1.18', '2008/02/20', '07:39:01',
'hewitson', 'm/sigproc/frequency_domain/', '9d47eb416fa21a099cb75512a291597d'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
frequency_domain/ltpda_rms.m', 'ltpda_rms', '1.8', '2007/11/08', '10:35:04',
'hewitson', 'm/sigproc/frequency_domain/', '6cae5cc2e6abff3f0e6da5e763c06749'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
frequency_domain/ltpda_tfe.m', 'ltpda_tfe', '1.14', '2007/07/16', '12:52:21',
'ingo', 'm/sigproc/frequency_domain/', 'c10ceed654995fa164054a4d88d0543a'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
frequency_domain/ltpda_xspec.m', 'ltpda_xspec', '1.10', '2007/11/08', '10:24:54',
'hewitson', 'm/sigproc/frequency_domain/', '73ce0fdb7232c520bb096cb4479e8063'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
time_domain/ltpda_arma_time.m', 'ltpda_arma_time', '1.1', '2008/02/15', '14:45:28',
'miquel', 'm/sigproc/time_domain/', '3231ca5e64cffd615284d2c5c81c72d1'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
time_domain/ltpda_chkts.m', 'ltpda_chkts', '1.4', '2007/11/07', '15:57:27', 'ingo',
'm/sigproc/time_domain/', '18c5f9ccd0379b2b2885b90e5e28ac4f'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
time_domain/ltpda_gapfilling.m', 'ltpda_gapfilling', '1.2', '2008/01/29',
'10:55:27', 'josep', 'm/sigproc/time_domain/',
'ce52aed820fa46f09cdf09d2ad216e26'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
time_domain/ltpda_polydetrend.m', 'ltpda_polydetrend', '1.8', '2008/02/08',
'14:59:31', 'anneke', 'm/sigproc/time_domain/',
```

```
'7d66ae1372701f474d1c577e1328b4da'; ...
0677         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
time_domain/ltpda_spikecleaning.m', 'ltpda_spikecleaning', '1.2', '2008/01/29',
'10:50:26', 'josep', 'm/sigproc/time_domain/',
'6e4add4f06c3ab0a6287fee9d07187ca'; ...
0678         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
time_domain/ltpda_timedomainfit.m', 'ltpda_timedomainfit', '1.5', '2007/11/08',
'10:35:04', 'hewitson', 'm/sigproc/time_domain/',
'b87fe86380d370e1c884fe4b6511f4f2'; ...
0679         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
time_domain/ltpda_xcorr.m', 'ltpda_xcorr', '1.4', '2007/11/08', '10:35:04',
'hewitson', 'm/sigproc/time_domain/', '4c76ff736c77bff41a6a6b1a9a1a1cd9'; ...
0680         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/sigproc/
time_domain/polydetrend.m', 'polydetrend', '1.2', '2007/07/16', '12:52:21', 'ingo',
'm/sigproc/time_domain/', '03f699096d5473174a9d51ea6441f7bb'; ...
0681         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
ltpda_gps2utc.m', 'ltpda_gps2utc', '1.4', '2007/07/27', '09:31:22', 'ingo', 'm/
timetools/', 'afdbe79a311bc65547ae6c288bdcc947'; ...
0682         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
ltpda_reformat_date.m', 'ltpda_reformat_date', '1.1', '2007/07/25', '09:21:35',
'ingo', 'm/timetools/', 'ba88f5cf1c8b235f65968bc18dc7ef37'; ...
0683         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
ltpda_utc2gps.m', 'ltpda_utc2gps', '1.3', '2007/07/16', '12:52:21', 'ingo', 'm/
timetools/', 'e6c8bb653bdbed7a1609fcc69f227ae9'; ...
0684         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/ltpda_jacobian.m', 'ltpda_jacobian', '1.1', '2008/02/14',
'12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'c24617b02e46420a499d1f3cd02aea89'; ...
0685         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/ltpda_ss_assemble.m', 'ltpda_ss_assemble', '1.1', '2008/02/14',
'12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'9a043f202806226d3a499a2847ed255b'; ...
0686         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/ltpda_ss_assemble_arrange.m', 'ltpda_ss_assemble_arrange',
'1.1', '2008/02/14', '12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'e6b3838e7d81a9fc519396edbe093327'; ...
0687         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/ltpda_ss_assemble_fusion.m', 'ltpda_ss_assemble_fusion', '1.1',
'2008/02/14', '12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'fe511128abc6f090627e34786c72f7ff'; ...
0688         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/ltpda_ss_check.m', 'ltpda_ss_check', '1.1', '2008/02/14',
'12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'18e4a1ccd17103a83fef5dd9e1005bb6'; ...
0689         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/ltpda_ss_make.m', 'ltpda_ss_make', '1.1', '2008/02/14',
'12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'ea265f2162d5a74568c86514adb78941'; ...
0690         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/ltpda_ss_miir2ss.m', 'ltpda_ss_miir2ss', '1.1', '2008/02/14',
'12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'e698150267b78b8b83d0d5cf5c042ec6'; ...
0691         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
```

```
statespacefunctions/ltpda_ss_miir2ssVjordan.m', 'ltpda_ss_miir2ssVjordan', '1.1',
'2008/02/14', '12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'9338a63cf5476d9001346f081a029728'; ...
0692         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/ltpda_ss_modify.m', 'ltpda_ss_modify', '1.1', '2008/02/14',
'12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'd1dab64bd982927a6576883388336197'; ...
0693         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/ltpda_ss_modify_old.m', 'ltpda_ss_modify_old', '1.1',
'2008/02/14', '12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'915b57cc30e7aa8aad5573a3460c1585'; ...
0694         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/ltpda_ss_modify_time_setup.m', 'ltpda_ss_modify_time_setup',
'1.1', '2008/02/14', '12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'fee659f960e65cea582b89a83f52e8de'; ...
0695         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/ltpda_ss_pz2ss.m', 'ltpda_ss_pz2ss', '1.1', '2008/02/14',
'12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'b849dd40b105f08688f1d40e06acb749'; ...
0696         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/ltpda_ss_tomiir.m', 'ltpda_ss_tomiir', '1.1', '2008/02/14',
'12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'b9aa29f27033c6c204ead63fedf86c77'; ...
0697         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/utp_ltpda_ss_assemble_1.m', 'utp_ltpda_ss_assemble_1', '1.1',
'2008/02/14', '12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'542fd21475eea3caca013a6b5b966126'; ...
0698         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/utp_ltpda_ss_assemble_2.m', 'utp_ltpda_ss_assemble_2', '1.1',
'2008/02/14', '12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'd4ed71e8f04acd07d4ac9b301ae2ea7b'; ...
0699         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/utp_ltpda_ss_make.m', 'utp_ltpda_ss_make', '1.1', '2008/02/14',
'12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'7da3a1c3268c282d8635e0d6845b3878'; ...
0700         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/utp_ltpda_ss_miir2ss.m', 'utp_ltpda_ss_miir2ss', '1.1',
'2008/02/14', '12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'53019a6be92072be88ef51c1e0ca114f'; ...
0701         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/utp_ltpda_ss_modify.m', 'utp_ltpda_ss_modify', '1.1',
'2008/02/14', '12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'fb7041ddfc4d740bdd084b6e2de76bb7'; ...
0702         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/utp_ltpda_ss_modify_time_setup.m',
'utp_ltpda_ss_modify_time_setup', '1.1', '2008/02/14', '12:12:41', 'adrien', 'm/
timetools/statespacefunctions/', '39696cf61e84365ac5883f3f533d47c8'; ...
0703         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/utp_ltpda_ss_pz2ss.m', 'utp_ltpda_ss_pz2ss', '1.1',
'2008/02/14', '12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'01ad10e843136f284564f7ed8320d4e1'; ...
0704         '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/utp_ltpda_ss_random_generate.m', 'utp_ltpda_ss_random_generate',
```

```
'1.1', '2008/02/14', '12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'a5eaa77506caba862653d18b8f8f2fcc'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//m/timetools/
statespacefunctions/utp_ltpda_ss_tomiir.m', 'utp_ltpda_ss_tomiir', '1.1',
'2008/02/14', '12:12:41', 'adrien', 'm/timetools/statespacefunctions/',
'bbafcacf62414709208b35f4f9724bbb'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//mex/ltpda_dft.m',
'ltpda_dft', '1.1', '2008/01/15', '20:20:41', 'hewitson', 'mex/',
'e25c0cc5bbeb47d702e3b96587ddd483'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda//mex/ltpda_polyreg.
m', 'ltpda_polyreg', '1.1', '2008/02/06', '07:07:31', 'hewitson', 'mex/',
'eccb1ec65c73b6891ca288233525be09'; ...
        };
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

set

PURPOSE ^

SET sets an xyzdata property.

SYNOPSIS ^

function xyz = set(xyz, varargin)

DESCRIPTION ^

```
 SET sets an xyzdata property.

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

 DESCRIPTION: SET sets an xyzdata property.

 CALL: c = set(c, 'name',    'my_name');
       c = set(c, 'x',        X);
       c = set(c, 'x',       [1 2 3]);
       c = set(c, 'y',        Y);
       c = set(c, 'y',       [1 2 3]);
       c = set(c, 'z',        Z);
       c = set(c, 'z',       [1 2 3]);
       c = set(c, 'xunits',  's');
       c = set(c, 'yunits',  'Hz');
       c = set(c, 'zunits',  'V');
       c = set(c, 'created',  time());
       c = set(c, 'version', 'xyzdata/set.m,v 1.7...');

 VERSION: $Id: set.m,v 1.2 2008/02/12 15:20:33 ingo Exp $

 HISTORY: 24-12-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- display DISPLAY overloads display functionality for xyzdata objects.

This function is called by:

SOURCE CODE ^

```
function xyz = set(xyz, varargin)
% SET sets an xyzdata property.
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 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/display">timespan/display</a>  -  overloads display
functionality for timespan objects.
   <a href="matlab:help timespan/eq">timespan/eq</a>       -  overloads the ==
operator for timespan objects.
   <a href="matlab:help timespan/isfield">timespan/isfield</a>  -  tests if the
given field is one of the object properties.
   <a href="matlab:help timespan/ne">timespan/ne</a>       -  overloads the ~=
operator for timespan objects.
   <a href="matlab:help timespan/save">timespan/save</a>     -  a timespan object to
file.
   <a href="matlab:help timespan/set">timespan/set</a>      -  set timespan
properties.
   <a href="matlab:help timespan/subsref">timespan/subsref</a>  -  Define field name
indexing for timespan objects.
   <a href="matlab:help timespan/timespan">timespan/timespan</a> -  timespan object
class constructor.
   <a href="matlab:help timespan/xml">timespan/xml</a>      -  convert a time object
to an DOM XML object.
   <a href="matlab:help timespan/xmladd">timespan/xmladd</a>   -  add a timespan
object to the given node of the input XML DOM
   <a href="matlab:help timespan/xmlparse">timespan/xmlparse</a> -  creates a
timespan from a timespan xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 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

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_xaxis

PURPOSE ^

LTPDA_XAXIS Set the X axis range of the current figure

SYNOPSIS ^

function ltpda_xaxis(x1,x2)

DESCRIPTION ^

```
 LTPDA_XAXIS Set the X axis range of the current figure

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

 DESCRIPTION: LTPDA_XAXIS Set the X axis range of the current figure

 CALL:        ltpda_xaxis(x1,x2)

 VERSION:     $Id: ltpda_xaxis.m,v 1.4 2007/07/16 12:52:20 ingo Exp $

 HISTORY:     09-10-2001 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_xaxis(x1,x2)
% LTPDA_XAXIS Set the X axis range of the current figure
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: LTPDA_XAXIS Set the X axis range of the current figure
%
% CALL:        ltpda_xaxis(x1,x2)
%
```

```
% VERSION:     $Id: ltpda_xaxis.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
%
% HISTORY:     09-10-2001 M Hewitson
%                 Creation
%
%%%
```

set

PURPOSE ^

SET set a zero property.

SYNOPSIS ^

function p = set(p, varargin)

DESCRIPTION ^

```
 SET set a zero property.

%%%
```

```
function p = set(p, varargin)
% SET set a zero property.
%
%%%
```

```
  elseif strcmp(in, 'Category')
    p = CATEGORY;
    return
  end
end

if nargin >= 2 && isa(propArgin{1}, 'plist')
  pl         = propArgin{1};
  propArgin = {};
  for ii = 1:nparams(pl)
    %% The 'key' name of a plist/param is always stored with upper case
    %% characters. To set a ao property with this key we have to lower the
    %% 'key' characters.
    propArgin{end+1} = lower(pl.params(ii).key);
    propArgin{end+1} =       pl.params(ii).val;
  end
end

while length(propArgin) >= 2

  prop       = propArgin{1};
  val        = propArgin{2};
  propArgin = propArgin(3:end);

  %%% Special case: If propName is equal to 'f' then we have to compute 'ri'
  if strcmp(prop, 'f')
    for jj = 1:numel(p)
      p(jj).f  = val;
      p(jj).ri = pfq2ri(p(jj).f, p(jj).q);
    end

  %%% Special case: If propName is equal to 'q' then we have to compute 'ri'
  elseif strcmp(prop, 'q')
    for jj = 1:numel(p)
      p(jj).q  = val;
      p(jj).ri = pfq2ri(p(jj).f, p(jj).q);
    end

  %%% Special case: If propName is equal to 'ri' then we have to compute 'q' and 'f'
  elseif strcmp(prop, 'ri')
    if size(val,2) > size(val,1), val = val.'; end
    for jj = 1:numel(p)
      if length(val) == 1
        p(jj).ri = [val; conj(val)];
      else
        p(jj).ri = val;
      end
      [p(jj).f, p(jj).q] = pri2fq(p(jj).ri);
    end

  %%% Normal case
```

```
  elseif ismember(prop, zero_fields)
    for jj = 1:numel(p)
      p(jj).(prop) = val;
    end
  else
    error('### ''%s'' is not a valid %s-object property.', prop, class(p));
  end

end

if (nargout == 0)
  display(p);
  error('\n### No output variable!\n### Please use: %s = set(%s, ''key'', ''value'');', inputname(1), inputname(1));
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: timespan/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%   class: timespan/private   %%%%%%%%%%%%%%%%%%

   <a href="matlab:help timespan/private/compute_interval">timespan/private/
compute_interval</a> -  compute the interval of the time span.
   <a href="matlab:help timespan/private/fromxml">timespan/private/fromxml</
a>          -  read from part of an XML tree and create a timespan object.
   <a href="matlab:help timespan/private/retrieve">timespan/private/retrieve</
a>         -  timespan objects with given ids from the LTPDA repository specified by
the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 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

 VERSION:     $Id: getNodes.m,v 1.13 2008/02/22 20:35:33 hewitson Exp $

 HISTORY:     02-02-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- get GET get history object properties.
- getNodes GETNODES converts a history object to a nodes structure suitable for plotting as a tree.

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.
- plot PLOT plots a history object as a tree diagram.

SOURCE CODE ^

```
function [varargout] = getNodes(varargin)
% GETNODES converts a history object to a nodes structure suitable for plotting as a tree.
%
%%%
```

```
% Set input
h     = varargin{1};
n     = varargin{2};
pn    = varargin{3};
a     = varargin{4};
nodes = varargin{5};

% set my node to next number
nn = h.n;
if nn<0
  h.n = a;
end

% set my parent node
nn = h.pn;
if nn<0
  h.pn = pn;
  n = [n pn];
  pl = h.plist;
  nodes(a).pn     = pn;
  nodes(a).invars = h.invars;
  nodes(a).n      = h.n;
  nodes(a).names  = h.name;
  nodes(a).params = char(pl);
  nodes(a).pl     = pl;
  nodes(a).h      = h;
  disp(sprintf(' -- scanning node %d [%s]', pn, char(nodes(a).names)));
end

% I need my parent node
pn = h.n;

% get my children nodes
ihs = h.inhists;

% Now decide what to do with my children
if isa(ihs, 'history')
  for i=1:length(ihs)
    % get child number
    nn = ihs(i).n;
    % if this child is not set
    if nn < 0
      % set it
      a = a + 1;
      [n,a, nodes] = getNodes(ihs(i), n, pn, a, nodes);
    else % else go back to my parent
      [n,a, nodes] = getNodes(h, n, get(h,'pn'), a, nodes);
    end
  end
end

% Set outputs
```

```
varargout{1} = n;
varargout{2} = a;
varargout{3} = nodes;
```

Generated on Tue 26-Feb-2008 10:52:52 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.
```

```
               The following call returns a parameter list object that contains
               the default parameter values:

               >> pl = ltpda_xcorr('Params')

 VERSION:      $Id: ltpda_xcorr.m,v 1.4 2007/11/08 10:35:04 hewitson Exp $

 HISTORY:      07-02-2007 M Hewitson
                  Creation

%%%
```

```
%               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.
%
%              The following call returns a parameter list object that contains
%              the default parameter values:
%
%              >> pl = ltpda_xcorr('Params')
%
% VERSION:     $Id: ltpda_xcorr.m,v 1.4 2007/11/08 10:35:04 hewitson Exp $
%
% HISTORY:     07-02-2007 M Hewitson
%                 Creation
%
%%%
```

```

% check plist
if isempty(ps)
  pl = getDefaultPL();
else
  pl = combine(ps, getDefaultPL);
end

na = length(as);
if na < 2
  error('### LTPDA_XSPEC needs at least two AOs to make a transfer function.');
end


%---------------- Resample all AOs

disp('*** Resampling AOs...');
fsmax = findFsMax(as);
fspl  = plist(param('fsout', fsmax));
bs = [];
for i=1:na
  a = as(i);
  % check this is a time-series object
  if ~isa(a.data, 'tsdata')
    error('### ltpda_xspec requires tsdata (time-series) inputs.');
  end
  % Check Fs
  if a.data.fs ~= fsmax
    warning(sprintf('!!! Resampling AO %s/%s to %f Hz', a.name, a.data.name, fsmax));
    a = resample(a, fspl);
  end
  bs = [bs a];
end
as = bs;
clear bs;
disp('*** Done.');


%---------------- Truncate all vectors

% Get shortest vector
disp('*** Truncating all vectors...');
lmin = findShortestVector(as);
nsecs = lmin / fsmax;
bs = [];
for i=1:na
  a = as(i);
  if len(a) ~= lmin
    warning(sprintf('!!! Truncating AO %s/%s to %d secs', a.name, a.data.name, nsecs));
    bs = [bs  select(a, 1:lmin)];
```

```
  else
    bs = [bs a];
  end
end
as = bs;
clear bs;
disp('*** Done.');

%----------------- check input parameters

% Maximum lag for Xcorr
MaxLag = find(pl, 'MaxLag');

% Scale for Xcorr
scale = find(pl, 'Scale');

% Loop over input AOs
bs = cell(na);
for j=1:na
  aj = as(j);
  for k=1:na
    ak = as(k);
    % -------- Make Xspec estimate

    % Compute cross-correlation estimates using XCORR
    if MaxLag == -1
      MaxLag = len(aj);
    end
    [c,lags] = xcorr(aj.data.y,ak.data.y, MaxLag, scale);

    % create new output fsdata
    xy = xydata(lags./fsmax, c);
    xy = set(xy, 'name', sprintf('xcorr(%s,%s)', aj.data.name, ak.data.name));
    xy = set(xy, 'xunits', 's');
    xy = set(xy, 'yunits', '');

    %----------- 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
      h = history(ALGONAME, VERSION, pl, [aj.hist ak.hist]);
    else
      h = history(ALGONAME, VERSION, pl, [ak.hist aj.hist]);
    end
    h = set(h, 'invars', [invars(j) invars(k)]);

    % make output analysis object
    b = ao(xy, h);

    % set name
```

```
    if isempty(invars{k})
      nk = ak.name;
    else
      nk = invars{k};
    end
    if isempty(invars{j})
      nj = aj.name;
    else
      nj = invars{j};
    end
    b = setnh(b, 'name', sprintf('xcorr(%s->%s)', nj, nk));

    % add to outputs
    bs(j,k) = {b};
  end % End second loop over AOs
end % End first loop over AOs

% now we have a cell matrix of AOs but
% we want a normal matrix
b  = [bs{:}];
varargout{1} = reshape(b, na, na);


%-----------------------------------------------------------------------
% 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


%-----------------------------------------------------------------------
% 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

%-----------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

disp('* creating default plist...');
```

```
plo = plist();
plo = append(plo, param('MaxLag', -1));
plo = append(plo, param('Scale', 'none'));
disp('* done.');
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

setFreq

PURPOSE ^

SETFREQ sets the frequency vector of a fsdata object based on the length of the y vector and the sample rate.

SYNOPSIS ^

function fsd = setFreq(fsd, varargin)

DESCRIPTION ^

```
 SETFREQ sets the frequency vector of a fsdata object based on the length of the y
vector and the sample rate.

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

 DESCRIPTION: SETFREQ sets the frequency vector of a fsdata object based on the
              length of the y vector and the sample rate.

 CALL:        fsd = setFreq(fsd)

 VERSION:     $Id: setFreq.m,v 1.5 2008/02/12 22:11:21 hewitson Exp $

 HISTORY:     31-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- reshapeF RESHAPEF reshape the frequency vector to match the y vector.

This function is called by:

- fsdata FSDATA frequency-series object class constructor.

SOURCE CODE ^

```
function fsd = setFreq(fsd, varargin)
% SETFREQ sets the frequency vector of a fsdata object based on the length of
the y vector and the sample rate.
```

```
%
%%%
```

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: tsdata %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: tsdata   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help tsdata/char">tsdata/char</a>             -  convert a param
object into a string.
   <a href="matlab:help tsdata/display">tsdata/display</a>          -  overloads
display functionality for tsdata objects.
   <a href="matlab:help tsdata/eq">tsdata/eq</a>               -  overloads the ==
operator for tsdata objects.
   <a href="matlab:help tsdata/get">tsdata/get</a>              -  get tsdata
properties.
   <a href="matlab:help tsdata/get_xy_values">tsdata/get_xy_values</a>    -  returns
the x-axis and/or y-axis values.
   <a href="matlab:help tsdata/isfield">tsdata/isfield</a>          -  tests if the
given field is one of the object properties.
   <a href="matlab:help tsdata/ne">tsdata/ne</a>               -  overloads the ~=
operator for tsdata objects.
   <a href="matlab:help tsdata/reshapeT">tsdata/reshapeT</a>         -  reshape the
time-vector to match the x vector.
   <a href="matlab:help tsdata/save">tsdata/save</a>             -  a tsdata object
to file.
   <a href="matlab:help tsdata/set">tsdata/set</a>              -  sets a tsdata
property.
   <a href="matlab:help tsdata/setTime">tsdata/setTime</a>          -  sets the time-
vector of a tsdata object based on the length of the x vector and the sample rate.
   <a href="matlab:help tsdata/set_xy_axis">tsdata/set_xy_axis</a>      -  set the x-
axis and y-axis values.
   <a href="matlab:help tsdata/single_operation">tsdata/single_operation</a> -
implements specified operator overload for tsdata objects.
   <a href="matlab:help tsdata/subsasgn">tsdata/subsasgn</a>         -  define index
assignment for for tsdata properties.
   <a href="matlab:help tsdata/subsref">tsdata/subsref</a>          -  Define field
name indexing for tsdata objects.
   <a href="matlab:help tsdata/tsdata">tsdata/tsdata</a>           -  time-series
object class constructor.
   <a href="matlab:help tsdata/xml">tsdata/xml</a>              -  convert a tsdata
```

```
object to an DOM XML object.
   <a href="matlab:help tsdata/xmladd">tsdata/xmladd</a>           -  add a tsdata
object to the given node of the input XML DOM
   <a href="matlab:help tsdata/xmlparse">tsdata/xmlparse</a>         -  creates a
tsdata from a tsdata xml file.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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(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.

 VERSION:     $Id: getNodes_plot.m,v 1.5 2008/02/22 20:35:33 hewitson Exp $

 HISTORY:     14-06-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- get GET get history object properties.
- getNodes_plot GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.

This function is called by:

- getNodes_plot GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.
- plot PLOT plots a history object as a tree diagram.

SOURCE CODE ^

```
function [varargout] = getNodes_plot(varargin)
% GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.
%
%%%
```

```
      return
    elseif strcmp(in, 'Category')
      varargout{1} = CATEGORY;
      return
    end
  end
end

% Set input
h     = varargin{1};
n     = varargin{2};
pn    = varargin{3};
a     = varargin{4};
nodes = varargin{5};
depth = varargin{6};
MAX_DEPTH = varargin{7};

% set my node to next number
nn = h.n; %get(h,'n');
if nn<0
  h.n = a; % = set(h, 'n', a);
end

% set my parent node
nn = h.pn; %get(h,'pn');
if nn<0
  h.pn = pn; % = set(h, 'pn', pn);
  n = [n pn];
  nodes(a).pn     = pn;
  nodes(a).invars = h.invars; %get(h, 'invars');
  nodes(a).n      = h.n; %get(h, 'n');
  nodes(a).names  = h.name; %get(h, 'name');
  nodes(a).params = char(h.plist);
  nodes(a).pl     = h.plist;

  if depth == MAX_DEPTH
    nodes(a).names  = '...';
    nodes(a).params = '';
    nodes(a).invars = {};
  end

end

% I need my parent node
pn = h.n; %get(h, 'n');

% get my children nodes
ihs = h.inhists; %get(h, 'inhists');

% Now decide what to do with my children
if isa(ihs, 'history')
  for i=1:length(ihs)
```

```
    % get child number
    nn = ihs(i).n; %get(ihs(i), 'n');
    % if this child is not set
    if nn < 0
      if depth < MAX_DEPTH

        % set it
        a = a + 1;
        [n,a, nodes] = getNodes_plot(ihs(i), n, pn, a, nodes, depth+1, MAX_DEPTH);

      end
    else % else go back to my parent
      [n,a, nodes] = getNodes_plot(h, n, get(h,'pn'), a, nodes, depth, MAX_DEPTH);
    end
  end

end

% Set outputs
varargout{1} = n;
varargout{2} = a;
varargout{3} = nodes;
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_xml_read_datasamples

PURPOSE ^

LTPDA_XML_READ_DATASAMPLES read data samples from the given node of an XML file.

SYNOPSIS ^

function data = ltpda_xml_read_datasamples(node, dType)

DESCRIPTION ^

```
LTPDA_XML_READ_DATASAMPLES read data samples from the given node of an XML file.

M Hewitson 19-02-07

$Id: ltpda_xml_read_datasamples.m,v 1.5 2007/10/26 14:39:54 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function data = ltpda_xml_read_datasamples(node, dType)

% LTPDA_XML_READ_DATASAMPLES read data samples from the given node of an XML file.
%
% M Hewitson 19-02-07
%
% $Id: ltpda_xml_read_datasamples.m,v 1.5 2007/10/26 14:39:54 hewitson Exp $
%



switch dType
  case 'real'

    disp('-- reading real data samples');

```

```
    data.x = [];
    data.y = [];

    children = node.getChildNodes;
    for j=1:children.getLength
      ch = children.item(j-1);
      nodeName = char(ch.getNodeName);
      cont = char(ch.getTextContent);
      switch nodeName
        case 'xset'
          vals = sscanf(cont, '%g ');
          data.x = [data.x ; vals];
          disp(['* read ' num2str(length(data.x)) ' X samples']);
        case 'yset'
          vals = sscanf(cont, '%g ');
          data.y = [data.y  ; vals];
          disp(['* read ' num2str(length(data.y)) ' Y samples']);
        otherwise
          %       warning(['!!! unknown data sample child node.' nodeName]);
      end
    end

%     data.x = data.x';
%     data.y = data.y';

  case 'complex'

    disp('-- reading complex data samples');

    data.x = [];
    ry     = [];
    iy     = [];

    children = node.getChildNodes;
    for j=1:children.getLength
      ch = children.item(j-1);
      nodeName = char(ch.getNodeName);
      cont = char(ch.getTextContent);
      switch nodeName
        case 'xset'
          vals = sscanf(cont, '%f ');
          data.x = [data.x ; vals];
          disp(['* read ' num2str(length(data.x)) ' X samples']);
        case 'ryset'
          vals = sscanf(cont, '%f ');
          ry = [ry ; vals];
          disp(['* read ' num2str(length(ry)) ' real Y samples']);
        case 'iyset'
          vals = sscanf(cont, '%f ');
          iy = [iy ; vals];
          disp(['* read ' num2str(length(iy)) ' imag Y samples']);
```

```
        otherwise
          %       warning(['!!! unknown data sample child node.' nodeName]);
      end
    end

%     data.x = data.x';
%     ry     = ry';
%     iy     = iy';

    data.y = complex(ry, iy);

  otherwise
    error('### unknown data type.');
end


% END

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

setTime

PURPOSE ^

SETTIME sets the time-vector of a tsdata object based on the length of the x vector and the sample rate.

SYNOPSIS ^

function ts = setTime(ts, varargin)

DESCRIPTION ^

```
 SETTIME sets the time-vector of a tsdata object based on the length of the x vector
and the sample rate.

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

 DESCRIPTION: SETTIME sets the time-vector of a tsdata object based on the
              length of the x vector and the sample rate.

 CALL:        tsdata = setTime(tsdata)

 VERSION:     $Id: setTime.m,v 1.7 2008/02/13 12:12:49 mauro Exp $

 HISTORY:     31-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- reshapeT RESHAPET reshape the time-vector to match the x vector.

This function is called by:

- tsdata TSDATA time-series object class constructor.

SOURCE CODE ^

```
0001 function ts = setTime(ts, varargin)
0002 % SETTIME sets the time-vector of a tsdata object based on the length of the x
vector and the sample rate.
0003 %
```

```
%%%
```

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: tsdata/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: tsdata/private   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help tsdata/private/fromxml">tsdata/private/fromxml</a>  -  read
from part of an XML tree and create a tsdata object.
   <a href="matlab:help tsdata/private/retrieve">tsdata/private/retrieve</a> -
tsdata objects with given ids from the LTPDA repository specified by the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_xmlread

PURPOSE ^

LTPDA_XMLREAD Reads a XML object

SYNOPSIS ^

function values = ltpda_xmlread(node, obj_name)

DESCRIPTION ^

```
 LTPDA_XMLREAD Reads a XML object

%%%
```

```
                        - empty cell
                        - empty double
                        - empty char
                        - char
                        - double
                        - logical
                        - java (necessary for timezone)
                   --> <cell>      ...
                   --> <object>    ...
                   --> <real_data> ...
                       <imag_data> ...

                </property>
            OR  </cell>

                -------   real_data   --------|--------   imag_data   --------
                                               |
                <real_data>('type'-attribute) | <imag_data> ('type' -attribute)
                           ('shape'-attribute)|             ('shape'-attribute)
                                               |
                   --> <matrix> ...            |    --> <matrix> ...
                   --> <vector> ...            |    --> <vector> ...
                                               |
                </real_data>                   | </imag_data>
                                               |
                ---------   matrix   --------------------   vector   ---------
                                               |
                <matrix> ('type' -attribute)   | <vector> ('type' -attribute)
                                               |
                   row vector (double)         |    column vector (double)
                                               |
                </matrix>                      | </vector>

 SYMBOLS:       --> Marks a choice between alternatives.
                ... Indicate that an element may be repeated.

 VERSION:       $Id: ltpda_xmlread.m,v 1.11 2008/02/25 16:13:56 ingo Exp $

 HISTORY:       31-01-2008 Diepholz
                   Creation

 SEE ALSO:      ltpda_xmlwrite, ltpda_xmlread, ltpda_saveobj

%%%
```

- ltpda_xmlread LTPDA_XMLREAD Reads a XML object

SUBFUNCTIONS ^

- function values = xml_read_ltpda_object(node)
- function obj = xml_read_object(node)
- function prop_value = xml_read_property_cell(node)
- function value = xml_read_real_imag_data(node)
- function values = xml_read_atomic_element(node)
- function shape = getShape(node)
- function header_displayed = TerminalOutput(node, header_displayed, node_name, number)

SOURCE CODE ^

```
% LTPDA_XMLREAD Reads a XML object
%
%%%
```

```
%                         - empty cell
%                         - empty double
%                         - empty char
%                         - char
%                         - double
%                         - logical
%                         - java (necessary for timezone)
%                   --> <cell>      ...
%                   --> <object>    ...
%                   --> <real_data> ...
%                       <imag_data> ...
%
%                 </property>
%             OR  </cell>
%
%                 -------   real_data   --------|--------   imag_data   --------
%                                                |
%                 <real_data>('type'-attribute) | <imag_data> ('type' -attribute)
%                            ('shape'-attribute)|             ('shape'-attribute)
%                                                |
%                   --> <matrix> ...             |    --> <matrix> ...
%                   --> <vector> ...             |    --> <vector> ...
%                                                |
%                 </real_data>                   | </imag_data>
%                                                |
%                 ---------   matrix   --------------------   vector   ---------
%                                                |
%                 <matrix> ('type' -attribute)   | <vector> ('type' -attribute)
%                                                |
%                   row vector (double)          |    column vector (double)
%                                                |
%                 </matrix>                      | </vector>
%
% SYMBOLS:        --> Marks a choice between alternatives.
%                 ... Indicate that an element may be repeated.
%
% VERSION:        $Id: ltpda_xmlread.m,v 1.11 2008/02/25 16:13:56 ingo Exp $
%
% HISTORY:        31-01-2008 Diepholz
%                   Creation
%
% SEE ALSO:       ltpda_xmlwrite, ltpda_xmlread, ltpda_saveobj
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function values = ltpda_xmlread(node, obj_name)

  if nargin == 1
    node_name = char(node.getNodeName);

    switch node_name
      case 'ltpda_object'
        values = xml_read_ltpda_object(node);
      case 'object'
```

```
        values = xml_read_object(node);
      case 'property'
        values = xml_read_property_cell(node);
      case 'cell'
        values = xml_read_property_cell(node);
      case 'real_data'
        values = xml_read_real_imag_data(node);
      case 'imag_data'
        values = xml_read_real_imag_data(node);
      case 'matrix'
        %% row vector
        values = sscanf(char(child_node.getTextContent), '%g ', [1,inf]);
      case 'vector'
        %% column vector
        values = sscanf(char(child_node.getTextContent), '%g ');
      otherwise
        %% Search for the next valid node name.
        for ii = 1:node.getLength
          item = node.item(ii-1);
          if item.hasChildNodes
            values = ltpda_xmlread(item);
          end
        end
    end
  elseif nargin == 2

    values = [];

    query_node = node.getElementsByTagName('ltpda_object');

    for ii= 1:query_node.getLength

      node = query_node.item(ii-1);

      for jj = 1:node.getLength

        child_node = node.item(jj-1);

        if child_node.hasChildNodes

          % Get node name
          node_name = char(child_node.getNodeName);

          switch node_name
            case 'object'
              val =  xml_read_object(child_node);
              if strcmp(class(val), obj_name)
                values = [val values];
              else
                warning('### Skip the read object because it is from the class [%s] and not from the class [%s]', class(val), obj_name);
              end
```

```
            otherwise
          end
        end

      end % over all childs

    end % over all 'ltpda_objects'

  else
    error('### Invalid command of this function');
  end
end


%%%
```

```
        otherwise
          error('### The ''ltpda_object'' element can not contain a [%s] element.', node_name);
      end
    end
  end
end


%%%
```

```
              obj.(prop_name) = prop_value;
            catch
              warning('\n\n### Skip the unknown property [%s] in the object [%s]
\n', prop_name, class(obj));
              disp([char(10) 'Code me up to convert me into the new
structure !!!' char(10) ]);
            end
%           end

        otherwise
          error('### The ''object'' element can not contain a [%s] element.',
node_name);
      end
    end
  end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DESCRIPTION: XML_READ_PROPERTY_CELL Reads a property element or a cell element
%              with the form:
%
%              <property> ('type', 'prop_name' -attributes)
% OR:          <cell>     ('type'              -attribute)
%
%                --> atomic element
%                      - empty cell
%                      - empty double
%                      - empty char
%                      - char
%                      - double
%                      - logical
%                      - java (necessary for timezone)
%                --> <cell>      ...
%                --> <object>    ...
%                --> <real_data> ...
%                    <imag_data> ...
%
%              </property>
% OR:          </cell>
%
% SYMBOLS:     --> Marks a choice between alternatives.
%              ... Indicate that an element may be repeated.
%
% HISTORY:     31-01-2008 Diepholz
%                Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function prop_value = xml_read_property_cell(node)

  prop_value = [];
  real_value = [];
```

```
  imag_value = [];
  shape      = [];

  %% Is the length of the node is equal 1 or zero
  %% then is the property a atomic element
  if node.getLength == 0 || node.getLength == 1
    prop_value = xml_read_atomic_element(node);
    return
  end

  for ii = 1:node.getLength

    child_node = node.item(ii-1);

    if child_node.getNodeType == child_node.ELEMENT_NODE

      %% Get the node name
      node_name = char(child_node.getNodeName);

      switch node_name
        case 'cell'
          shape = getShape(node);
          prop_value{end+1} = xml_read_property_cell(child_node);
        case 'object'
          shape = getShape(child_node);
          prop_value = [prop_value xml_read_object(child_node)];
        case 'real_data'
          real_value = xml_read_real_imag_data(child_node);
        case 'imag_data'
          imag_value = xml_read_real_imag_data(child_node);
        otherwise
          error('### The ''property'' or ''cell'' element can not contain a [%s] element.', node_name);
      end

      %% Is real_value and imag_value filled then return the complex value
      if ~isempty(real_value) && ~isempty(imag_value)
        prop_value = complex(real_value, imag_value);
      %% Is real_value filled then return the read value
      elseif ~isempty(real_value)
        prop_value = real_value;
      end
    end
  end

  %% Reshape the prop_value if necessary
  if ~isempty(shape)
    prop_value = reshape(prop_value, shape);
  end

end
```

```
%%%
```

```
          if strcmp(node_type, 'sym')
            matrix_row = sym(char(child_node.getTextContent));
          else
            matrix_row = sscanf(char(child_node.getTextContent), '%g ', [1, inf]);
          end
          value = [value; matrix_row];
        case 'vector'
          %% column vector
          if strcmp(node_type, 'sym')
            vector_column = sym(char(child_node.getTextContent));
          else
            vector_column = sscanf(char(child_node.getTextContent), '%g ');
          end
          value = [value; vector_column];
          header_displayed = TerminalOutput(node, header_displayed, node_name, length(value));
        otherwise
          error('### The ''real_data'' or ''imag_data'' element can not contain a [%s] element.', node_name);
      end
    end
  end

  if strcmp(node_name, 'matrix')
    header_displayed = true;
    TerminalOutput(node, header_displayed, node_name, size(value,1));
  end
  value = reshape(value, shape);

end


%%%
```

```
  values = '';

  type = char(node.getAttribute('type'));
  switch type
    case {'', 'char'}
      values = char(node.getTextContent);

      %% Special case if 'values' is empty because
      %% there are more than an one empty char fields.
      %% For example: '' AND char('', '') AND char('', '', '') AND ...
      if isempty(values)
        shape = getShape(node);
        if any(shape)
          char_str = 'char(';
          for ii = 1:max(shape)
            char_str = [char_str, '''''', '];
          end
          char_str = char_str(1:end-2);
          cmd = strcat('values = ', char_str, ');');
          eval(cmd);
        else
          values = '';
        end
      end

    case 'double'
      number = char(node.getTextContent);

      %% Special case if 'values' is empty because
      %% there are more than an one empty double fields.
      %% For example: [] AND zeros(1,0) AND zeros(0,1) AND zeros(2,0) AND ...
      if isempty(number)
        shape = getShape(node);
        values = zeros(shape);
      else
        values = str2double(number);
      end

    case 'sym'
      number = char(node.getTextContent);

      %% Special case if 'values' is empty because
      %% there are more than an one empty double fields.
      %% For example: [] AND zeros(1,0) AND zeros(0,1) AND zeros(2,0) AND ...
      if isempty(number)
        shape = getShape(node);
        values = sym(shape);
      else
        values = sym(number);
      end

    case 'cell'
```

```
      cell_str = char(node.getTextContent);

      %% Special case if 'values' is empty because
      %% there are more than an one empty cell fields.
      %% For example: {} AND cell(1,0) AND cell(0,1) AND cell(2,0) AND ...
      if isempty(cell_str)
        shape = getShape(node);
        values = cell(shape);
      else
        error('### Should not happen because a not empty ''cell'' node have several child nodes.');
      end

    case 'logical'
      if strcmp (char(node.getTextContent), 'true')
        values = true;
      else
        values = false;
      end

    case 'sun.util.calendar.ZoneInfo'
      values = java.util.TimeZone.getTimeZone(char(node.getTextContent));
    otherwise
      error('### Unknown type attribute [%s].', type');
  end
end


%%%
```

```
% HISTORY:     31-01-2008 Diepholz
%                 Creation
%
%%%
```

```
0600
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: xydata %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: xydata   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help xydata/char">xydata/char</a>             -  convert a param
object into a string.
   <a href="matlab:help xydata/display">xydata/display</a>          -  overloads
display functionality for xydata objects.
   <a href="matlab:help xydata/eq">xydata/eq</a>               -  overloads the ==
operator for xydata objects.
   <a href="matlab:help xydata/get">xydata/get</a>              -  gets xydata
properties.
   <a href="matlab:help xydata/get_xy_values">xydata/get_xy_values</a>    -  returns
the x-axis and/or y-axis values.
   <a href="matlab:help xydata/isfield">xydata/isfield</a>          -  tests if the
given field is one of the object properties.
   <a href="matlab:help xydata/ne">xydata/ne</a>               -  overloads the ~=
operator for xydata objects.
   <a href="matlab:help xydata/reshapeX">xydata/reshapeX</a>         -  reshape the
X to match the Y vector.
   <a href="matlab:help xydata/save">xydata/save</a>             -  an xydata object
to file.
   <a href="matlab:help xydata/set">xydata/set</a>              -  sets a xydata
property.
   <a href="matlab:help xydata/setX">xydata/setX</a>             -  sets the time-
vector of a tsdata object based on the length of the x vector and the sample rate.
   <a href="matlab:help xydata/set_xy_axis">xydata/set_xy_axis</a>      -  set the x-
axis and y-axis values.
   <a href="matlab:help xydata/single_operation">xydata/single_operation</a> -
implements specified operator overload for xydata objects.
   <a href="matlab:help xydata/subsasgn">xydata/subsasgn</a>         -  define index
assignment for xydata properties.
   <a href="matlab:help xydata/subsref">xydata/subsref</a>          -  Define field
name indexing for xydata objects.
   <a href="matlab:help xydata/xml">xydata/xml</a>              -  convert an xydata
object to an DOM XML object.
   <a href="matlab:help xydata/xmladd">xydata/xmladd</a>           -  add a xydata
```

```
object to the given node of the input XML DOM
   <a href="matlab:help xydata/xmlparse">xydata/xmlparse</a>         -  creates a
xydata from a xydata xml file.
   <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 Tue 26-Feb-2008 10:52:52 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 1.2 2008/01/22 21:10:23 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 1.2 2008/01/22 21:10:23 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 ltpda_isuserobject(obj)
    objs = [objs ws_vars(j)];
  end

end



```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_xmlwrite

PURPOSE ^

LTPDA_XMLWRITE Add an object to a xml DOM project.

SYNOPSIS ^

function ltpda_xmlwrite(objs, xml, parent, property_name)

DESCRIPTION ^

```
 LTPDA_XMLWRITE Add an object to a xml DOM project.

%%%
```

```
                <property> ('type', 'prop_name' -attributes)
            OR  <cell>     ('type'              -attribute)

                   --> atomic element
                         - empty cell
                         - empty double
                         - empty char
                         - char
                         - double
                         - logical
                         - java (necessary for timezone)
                   --> <cell>      ...
                   --> <object>    ...
                   --> <real_data> ...
                       <imag_data> ...

                </property>
            OR  </cell>

                -------   real_data   --------|--------   imag_data   --------
                                               |
                <real_data>('type'-attribute) | <imag_data> ('type' -attribute)
                           ('shape'-attribute)|             ('shape'-attribute)
                                               |
                   --> <matrix> ...            |    --> <matrix> ...
                   --> <vector> ...            |    --> <vector> ...
                                               |
                </real_data>                   | </imag_data>
                                               |
                ---------   matrix   --------------------   vector   ---------
                                               |
                <matrix> ('type' -attribute)   | <vector> ('type' -attribute)
                                               |
                   row vector (double)         |    column vector (double)
                                               |
                </matrix>                      | </vector>

 SYMBOLS:       --> Marks a choice between alternatives.
                ... Indicate that an element may be repeated.

 VERSION:       $Id: ltpda_xmlwrite.m,v 1.8 2008/02/25 16:13:56 ingo Exp $

 HISTORY:       31-01-2008 Diepholz
                   Creation

 SEE ALSO:      ltpda_xmlwrite, ltpda_xmlread, ltpda_saveobj

%%%
```

- ltpda_num2str LTPDA_NUM2STR uses sprintf to convert a data vector to a string with a fixed precision.
- ltpda_xmlwrite LTPDA_XMLWRITE Add an object to a xml DOM project.

This function is called by:

- ltpda_saveobj SAVEOBJ saves an object to a file.
- ltpda_xmlwrite LTPDA_XMLWRITE Add an object to a xml DOM project.

SUBFUNCTIONS ^

- function xml_addvector(objs, xml, parent)
- function xml_addmatrix(objs, xml, parent)
- function header_displayed = TerminalOutput(parent, header_displayed, isreal_num, obj_type, number)

SOURCE CODE ^

```
% LTPDA_XMLWRITE Add an object to a xml DOM project.
%
%%%
```

```
%                -------------------------   property   -------------------------
%                --------------------------   cell   --------------------------
%
%                <property> ('type', 'prop_name' -attributes)
%            OR  <cell>     ('type'              -attribute)
%
%                   --> atomic element
%                         - empty cell
%                         - empty double
%                         - empty char
%                         - char
%                         - double
%                         - logical
%                         - java (necessary for timezone)
%                   --> <cell>      ...
%                   --> <object>    ...
%                   --> <real_data> ...
%                       <imag_data> ...
%
%                </property>
%            OR  </cell>
%
%                -------   real_data   --------|--------   imag_data   --------
%                                               |
%                <real_data>('type'-attribute) | <imag_data> ('type' -attribute)
%                           ('shape'-attribute)|             ('shape'-attribute)
%                                               |
%                   --> <matrix> ...            |    --> <matrix> ...
%                   --> <vector> ...            |    --> <vector> ...
%                                               |
%                </real_data>                   | </imag_data>
%                                               |
%                ---------   matrix   --------------------   vector   ---------
%                                               |
%                <matrix> ('type' -attribute)   | <vector> ('type' -attribute)
%                                               |
%                   row vector (double)         |    column vector (double)
%                                               |
%                </matrix>                      | </vector>
%
% SYMBOLS:       --> Marks a choice between alternatives.
%                ... Indicate that an element may be repeated.
%
% VERSION:       $Id: ltpda_xmlwrite.m,v 1.8 2008/02/25 16:13:56 ingo Exp $
%
% HISTORY:       31-01-2008 Diepholz
%                   Creation
%
% SEE ALSO:      ltpda_xmlwrite, ltpda_xmlread, ltpda_saveobj
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function ltpda_xmlwrite(objs, xml, parent, property_name)

```

```
%%%%%%   If the property_name is filled then create a new property node   %%%%%
if ~isempty(property_name)
  shape = sprintf('%dx%d', size(objs,1), size(objs,2));

  prop_node = xml.createElement('property');
  prop_node.setAttribute('prop_name', property_name);
  prop_node.setAttribute('shape', shape);
  prop_node.setAttribute('type', class(objs));
  parent.appendChild(prop_node);
  parent = prop_node;
end

%%%%%%%%%%%%%%%%   The object is a class object or a struct   %%%%%%%%%%%%%%%%%
if (isobject(objs) || isstruct(objs)) && ~isa(objs, 'sym')

  for ii = 1:numel(objs)

    obj = objs(ii);

    shape = sprintf('%dx%d', size(objs,1), size(objs,2));

    obj_node = xml.createElement('object');
    obj_node.setAttribute('type', class(obj));
    obj_node.setAttribute('shape', shape);

    parent.appendChild(obj_node);

    fields = fieldnames(obj);

    for jj = 1:length(fields)
      field = fields{jj};
      ltpda_xmlwrite(obj.(field), xml, obj_node, field);
    end
  end

%%%%%%%%%%%%%%%%%%%%%%%   The object is a java object   %%%%%%%%%%%%%%%%%%%%%%%
elseif isjava(objs)
  if strcmp(class(objs), 'sun.util.calendar.ZoneInfo')
    content = xml.createTextNode(char(objs.getID));
    parent.appendChild(content);
  else
    error('### Unknown Java');
  end

%%%%%%%%%%%%%%%%%%%%%%%   The object is a cell object   %%%%%%%%%%%%%%%%%%%%%%%
elseif iscell(objs)

  for ii = 1:numel(objs)

```

```
      obj = objs{ii};

      shape = sprintf('%dx%d', size(obj,1), size(obj,2));

      cell_node = xml.createElement('cell');
      cell_node.setAttribute('type', class(obj));
      cell_node.setAttribute('prop_name', property_name);
      cell_node.setAttribute('shape', shape);

      parent.appendChild(cell_node);

      ltpda_xmlwrite(obj, xml, cell_node, '');

    end

  %%%%%%%%%%%%%%%%%%%%   The object is a character string   %%%%%%%%%%%%%%%%%%%%
%
  elseif ischar(objs)
    content = xml.createTextNode(objs);
    parent.appendChild(content);

  %%%%%%%%%%%%%%%%%%%%%%%%%%   The object is a logical   %%%%%%%%%%%%%%%%%%%%%%%%%
%
  elseif islogical(objs)
    if numel(objs) == 1
      if objs
        content = xml.createTextNode('true');
      else
        content = xml.createTextNode('false');
      end
      parent.appendChild(content);
    else
      error ('### At the moment is only one logical allowed. The size is [%dx%
d]', size(objs,1), size(objs,2));
    end

  %%%%%%%%%%%%%%%%%%%%%%%%%%   The object is a number   %%%%%%%%%%%%%%%%%%%%%%%%%
%
  elseif isnumeric(objs) || isa(objs, 'sym')

    %%%%%   objs is a singel value   %%%%%
    if (size(objs,1) == 1) && (size(objs,2) == 1)
      if strcmp(class(objs), 'sym')
        number_str = char(objs, 20);
      else
        number_str = num2str(objs, 20);
      end
      content = xml.createTextNode(number_str);
      parent.appendChild(content);

      %%%%%   objs is a matrix   %%%%%
    elseif (size(objs,1) > 1) && (size(objs,2) > 1)

```

```
      xml_addmatrix(objs, xml, parent);

      %%%%%   objs is a vector   %%%%%
    elseif (size(objs,1) > 1) || (size(objs,2) > 1)

      xml_addvector(objs, xml, parent);

    end

  else
    error('### unknown type [%s]', class(objs));
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: XML_ADDVECTOR Add a vector element to the xml node. The element
%              have the form:
%
% REAL DATA:   <real_data type="vector">
%                 <vector type="double"> 1  2  3  4  5  6  7  8  9 10</vector>
%                 <vector type="double">11 12 13 14 15 16 17 18 19 20</vector>
%              </real_data>
%
% COMPLEX DATA:<real_data type="vector">
%                 <vector type="double"> 1  2  3  4  5  6  7  8  9 10</vector>
%                 <vector type="double">11 12 13 14 15 16 17 18 19 20</vector>
%              </real_data>
%
%              <imag_data type="vector">
%                 <vector type="double"> 1  2  3  4  5  6  7  8  9 10</vector>
%                 <vector type="double">11 12 13 14 15 16 17 18 19 20</vector>
%              </imag_data>
%
%              The vector objs will be split into several parts dependent from
%              the maximum size in one vector element.
%              Each part creates its own vectror element.
%
% HISTORY:     31-01-2008 Diepholz
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function xml_addvector(objs, xml, parent)

  n_min            = getappdata(0, 'xmlsetsize');
  header_displayed = true;

  shape = sprintf('%dx%d', size(objs,1), size(objs,2));

  %%%%%   Real data   %%%%%
  node_real = xml.createElement('real_data');
  node_real.setAttribute('type', 'vector');
```

```
  node_real.setAttribute('shape', shape);
  %% Set the parent attribute 'type' to vector
  parent.setAttribute('type', 'vector');
  parent.appendChild(node_real);

  write_number = real(objs);
  sample_num   = 0;

  while ~isempty(write_number)

    n = min(n_min, length(write_number));

    sample_num = sample_num + n;
    header_displayed = TerminalOutput(parent, header_displayed, true, 'vector', sample_num);

    item = xml.createElement('vector');
    item.setAttribute('type', class(objs));

    if strcmp(class(write_number), 'sym')
      number_str = char(write_number(1:n));
    else
      number_str = ltpda_num2str(write_number(1:n));
    end

    content = xml.createTextNode(number_str);
    node_real.appendChild(item);
    item.appendChild(content);

    write_number = write_number(n+1:end);
  end

  %%%%%   Imaginary data   %%%%%
  if ~isreal(objs)
    node_imag = xml.createElement('imag_data');
    node_imag.setAttribute('type', 'vector')
    node_imag.setAttribute('shape', shape);
    parent.appendChild(node_imag);

    write_number = imag(objs);
    sample_num   = 0;

    while ~isempty(write_number)
      n = min(n_min, length(write_number));

      sample_num = sample_num + n;
      header_displayed = TerminalOutput(parent, header_displayed, false, 'vector', sample_num);

      item = xml.createElement('vector');
      item.setAttribute('type', class(objs));

```

```
      if strcmp(class(write_number), 'sym')
        number_str = char(write_number(1:n));
      else
        number_str = ltpda_num2str(write_number(1:n));
      end

      content = xml.createTextNode(number_str);
      node_imag.appendChild(item);
      item.appendChild(content);

      write_number = write_number(n+1:end);
    end
  end
end

%%%
```

```
node_real.setAttribute('type', 'matrix');
node_real.setAttribute('shape', shape);
%% Set the parent attribute 'type' to matrix
parent.setAttribute('type', 'matrix');
parent.appendChild(node_real);

write_number = real(objs);

for ii = 1:size(write_number,1)

  item = xml.createElement('matrix');
  item.setAttribute('type', class(objs));

  if strcmp(class(write_number), 'sym')
    number_str = char(write_number(ii,:));
  else
    number_str = ltpda_num2str(write_number(ii,:));
  end

  content = xml.createTextNode(number_str);
  node_real.appendChild(item);
  item.appendChild(content);
end

TerminalOutput(parent, header_displayed, true, 'matrix', ii);

%%%%%   Imaginary data   %%%%%
if ~isreal(objs)
  node_imag = xml.createElement('imag_data');
  node_imag.setAttribute('type', 'matrix');
  node_imag.setAttribute('shape', shape);
  parent.appendChild(node_imag);

  write_number = imag(objs);

  for ii = 1:size(write_number,1)

    item = xml.createElement('matrix');
    item.setAttribute('type', class(objs));

    if strcmp(class(write_number), 'sym')
      number_str = char(write_number(ii,:));
    else
      number_str = ltpda_num2str(write_number(ii,:));
    end

    content = xml.createTextNode(number_str);
    node_imag.appendChild(item);
    item.appendChild(content);
  end
  TerminalOutput(parent, header_displayed, false, 'matrix', ii);
end
```

```
end

%%%
```

Generated on Tue 26-Feb-2008 10:52:52 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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: xydata/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: xydata/private   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help xydata/private/fromxml">xydata/private/fromxml</a>  -  read
from part of an XML tree and create a xydata object.
   <a href="matlab:help xydata/private/retrieve">xydata/private/retrieve</a> -
xydata objects with given ids from the LTPDA repository specified by the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get_curr_m_file_path

PURPOSE ^

GET_CURR_M_FILE_PATH returns the path for a mfile.

SYNOPSIS ^

function pth = get_curr_m_file_path (m_file_name)

DESCRIPTION ^

```
 GET_CURR_M_FILE_PATH returns the path for a mfile.

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

 DESCRIPTION: GET_CURR_M_FILE_PATH returns the path for a mfile.

 CALL:        path = get_curr_m_file_path(mfilename)
              path = get_curr_m_file_path('explore_ao')

 VERSION:     $Id: get_curr_m_file_path.m,v 1.2 2007/06/25 11:21:48 ingo Exp $

 HISTORY: 25-06-07  Diepholz
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function pth = get_curr_m_file_path (m_file_name)
% GET_CURR_M_FILE_PATH returns the path for a mfile.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: GET_CURR_M_FILE_PATH returns the path for a mfile.
%
% CALL:        path = get_curr_m_file_path(mfilename)
```

```
%              path = get_curr_m_file_path('explore_ao')
%
% VERSION:     $Id: get_curr_m_file_path.m,v 1.2 2007/06/25 11:21:48 ingo Exp $
%
% HISTORY: 25-06-07  Diepholz
%             Creation
%
%%%
```

ltpda_xscale

PURPOSE ^

LTPDA_XSCALE Set the X scale of the current axis

SYNOPSIS ^

function ltpda_xscale(scale)

DESCRIPTION ^

```
 LTPDA_XSCALE Set the X scale of the current axis

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

 DESCRIPTION: LTPDA_XSCALE Set the X scale of the current axis

 CALL:        ltpda_xscale('scale')      scale = 'lin' or 'log';

 VERSION:     $Id: ltpda_xscale.m,v 1.2 2007/07/16 12:52:20 ingo Exp $

 HISTORY:     02-05-2004 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_xscale(scale)
% LTPDA_XSCALE Set the X scale of the current axis
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DESCRIPTION: LTPDA_XSCALE Set the X scale of the current axis
%
% CALL:        ltpda_xscale('scale')      scale = 'lin' or 'log';
%
```

```
% VERSION:     $Id: ltpda_xscale.m,v 1.2 2007/07/16 12:52:20 ingo Exp $
%
% HISTORY:     02-05-2004 M Hewitson
%                 Creation
%
%%%
```

set_xy_axis

PURPOSE ^

SET_XY_AXIS set the x-axis and y-axis values.

SYNOPSIS ^

function data_out = set_xy_axis(data, x, y)

DESCRIPTION ^

```
 SET_XY_AXIS set the x-axis and y-axis values.

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

 DESCRIPTION: SET_XY_AXIS set the x-axis and y-axis values.

 CALL:       data = set_xy_axis (data,    x, y);
          ao.data = set_xy_axis (ao.data, x, y);

 VERSION: $Id: set_xy_axis.m,v 1.4 2008/02/12 19:54:05 hewitson Exp $

 HISTORY: 21-06-2007 Diepholz
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a cdata object into a string.

This function is called by:

SOURCE CODE ^

```
function data_out = set_xy_axis(data, x, y)
% SET_XY_AXIS set the x-axis and y-axis values.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SET_XY_AXIS set the x-axis and y-axis values.
```

```
%
% CALL:       data = set_xy_axis (data,    x, y);
%          ao.data = set_xy_axis (ao.data, x, y);
%
% VERSION: $Id: set_xy_axis.m,v 1.4 2008/02/12 19:54:05 hewitson Exp $
%
% HISTORY: 21-06-2007 Diepholz
%             Creation
%
%%%
```

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: zero %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: zero   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help zero/cat">zero/cat</a>      -  concatonate zeros into a
vector.
   <a href="matlab:help zero/char">zero/char</a>     -  convert a zero object into a
string.
   <a href="matlab:help zero/cz2iir">zero/cz2iir</a>   -  return a,b IIR filter
coefficients for a complex zero designed using the bilinear transform.
   <a href="matlab:help zero/display">zero/display</a>  -  overloads display
functionality for zero objects.
   <a href="matlab:help zero/eq">zero/eq</a>       -  overloads the == operator for
zero objects.
   <a href="matlab:help zero/get">zero/get</a>      -  get zero properties.
   <a href="matlab:help zero/isfield">zero/isfield</a>  -  tests if the given field
is one of the object properties.
   <a href="matlab:help zero/ne">zero/ne</a>       -  overloads the ~= operator for
zero objects.
   <a href="matlab:help zero/resp">zero/resp</a>     -  returns the complex response
of the zero object.
   <a href="matlab:help zero/rz2iir">zero/rz2iir</a>   -  Return a,b IIR filter
coefficients for a real zero designed using the bilinear transform.
   <a href="matlab:help zero/save">zero/save</a>     -  a zero object to file.
   <a href="matlab:help zero/set">zero/set</a>      -  set a zero property.
   <a href="matlab:help zero/string">zero/string</a>   -  writes a command string
that can be used to recreate the input zero object.
   <a href="matlab:help zero/subsref">zero/subsref</a>  -  Define field name
indexing for zero objects.
   <a href="matlab:help zero/xml">zero/xml</a>      -  convert a zero object to an
DOM XML object.
   <a href="matlab:help zero/xmladd">zero/xmladd</a>   -  add a zero object to the
given node of the input XML DOM
   <a href="matlab:help zero/xmlparse">zero/xmlparse</a> -  creates a zero from a
zero xml file.
   <a href="matlab:help zero/zero">zero/zero</a>     -  construct a pole object.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 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.

M Hewitson 18-05-07

$Id: get_window.m,v 1.5 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

- 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

This function is called by:

SUBFUNCTIONS ^

- function w = get_kaiser_window(w, N, psll)

SOURCE CODE ^

```
function w = get_window(varargin)

% GET_WINDOW returns the required window function as a structure.
%
% M Hewitson 18-05-07
%
```

```
% $Id: get_window.m,v 1.5 2008/02/13 18:38:02 ingo Exp $
%

PSLL = [];

% Get inputs
if nargin == 3
  w    = varargin{1};
  name = varargin{2};
  N    = varargin{3};
elseif nargin == 4
  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(''Types'') to see all supported windows.', win_name(5:end));
  end

else

  % Deal with Kaiser
  w = get_kaiser_window(w, N, PSLL);

end

%-----------------------------------------------------------------------
% Make Kaiser window
%
function w = get_kaiser_window(w, N, psll)

% Make window struct
w.name     = 'Kaiser';
w.alpha    = kaiser_alpha(psll);
w.psll     = psll;
w.rov      = kaiser_rov(w.alpha);
w.nenbw    = kaiser_nenbw(w.alpha);
w.w3db     = kaiser_w3db(w.alpha);
w.flatness = 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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_xspec

PURPOSE ^

LTPDA_XSPEC performs cross-spectral analysis of various forms.

SYNOPSIS ^

function varargout = ltpda_xspec(varargin)

DESCRIPTION ^

```
 LTPDA_XSPEC performs cross-spectral analysis of various forms.

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

 DESCRIPTION: LTPDA_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 = ltpda_xspec(a, pl, method, iALGO, iVER, invars);

 INPUTS:      a      - vector of input AOs
              pl     - input parameter list
              method - one of 'TF', 'CPSD', 'COHERE'
              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: ltpda_xspec.m,v 1.10 2007/11/08 10:24:54 hewitson Exp $

 HISTORY:     30-05-2007 M Hewitson
                 Creation

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = ltpda_xspec('Params')

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- 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_tfe LTPDA_TFE makes transfer function estimates of the time-series objects.

SUBFUNCTIONS ^

- function lmin = findShortestVector(as)
- function fs = findFsMax(as)

SOURCE CODE ^

```
function varargout = ltpda_xspec(varargin)
% LTPDA_XSPEC performs cross-spectral analysis of various forms.
%
%%%
```

```
pl     = varargin{2};
method = varargin{3};
iALGO  = varargin{4};
iVER   = varargin{5};
invars = varargin{6};

ALGONAME = iALGO;
VERSION  = [iVER '/' '$Id: ltpda_xspec.m,v 1.10 2007/11/08 10:24:54 hewitson Exp $'];


% initialise output array
bs = [];

na = length(as);
if na < 2
  error('### LTPDA_XSPEC needs at least two AOs to make a transfer function.');
end


%---------------- Resample all AOs

disp('*** Resampling AOs...');
fsmax = findFsMax(as);
fspl  = plist(param('fsout', fsmax));
bs = [];
for i=1:na
  a = as(i);
  % check this is a time-series object
  if ~isa(a.data, 'tsdata')
    error('### ltpda_xspec requires tsdata (time-series) inputs.');
  end
  % Check Fs
  if a.data.fs ~= fsmax
    warning(sprintf('!!! Resampling AO %s/%s to %f Hz', a.name, a.data.name, fsmax));
    a = resample(a, fspl);
  end
  bs = [bs a];
end
as = bs;
clear bs;
disp('*** Done.');


%---------------- Truncate all vectors

% Get shortest vector
disp('*** Truncating all vectors...');
lmin = findShortestVector(as);
nsecs = lmin / fsmax;
bs = [];
for i=1:na
```

```
  a = as(i);
  if len(a) ~= lmin
    warning(sprintf('!!! Truncating AO %s/%s to %d secs', a.name, a.data.name, nsecs));
    bs = [bs  select(a, 1:lmin)];
  else
    bs = [bs a];
  end
end
as = bs;
clear bs;
disp('*** Done.');

%----------------- check input parameters

% points in FFT
Nfft  = find(pl, 'Nfft');
if ~isempty(Nfft)
  if Nfft < 0
    Nfft = abs(Nfft) * lmin;
    disp(sprintf('! Using default Nfft of %g', Nfft))
    pl = pset(pl, 'Nfft', Nfft);
  end
end

% Window object
Win   = find(pl, 'Win');
if ~isempty(Win)
  if length(Win.win)~=Nfft
    switch Win.name
      case 'Kaiser'
        Win = specwin(Win.name, Nfft, Win.psll);
      otherwise
        Win = specwin(Win.name, Nfft);
    end
    disp(sprintf('! Reset window to %s(%d)', strrep(Win.name, '_', '\_'), length(Win.win)))
    pl = pset(pl, 'Win', Win);
  end
end

% desired overlap
Nolap = find(pl, 'Nolap');
if ~isempty(Nolap)
  if Nolap < 0
    % use window function rov
    Nolap = Win.rov/100;
    disp(sprintf('! Using recommended overlap of %d', Nolap))
    pl = pset(pl, 'Nolap', Nolap);
  end
end

% desired detrending order
```

```
order = find(pl, 'Order');
if isempty(order)
  order = 0;
  warning('! using default detrending order 0 (mean)');
  pl = append(pl, 'Order', order);
end

% Loop over input AOs
bs = cell(na);
for j=1:na
  aj = as(j);
  for k=1:na
    ak = as(k);
    % -------- Make Xspec estimate

    % Do detrending
    x = aj.data.y;
    y = ak.data.y;
    switch order
      case -1
        % do nothing
      case 0
        x = x - mean(x);
        y = y - mean(y);
      case 1
        x = detrend(x);
        y = detrend(y);
      otherwise
        y = polydetrend(y, order);
        y = polydetrend(y, order);
    end

    switch method
      case 'TF'
        % Compute TF using tfestimate
        [txy,f] = tfestimate(x,y,Win.win,round(Nolap*Nfft),Nfft,fsmax);
      case 'COHERE'
        % Compute coherence using mscohere
        [txy,f] = mscohere(x,y,Win.win,round(Nolap*Nfft),Nfft,fsmax);
      case 'CPSD'
        % Compute Cross-spectral density using CPSD
        [txy,f] = cpsd(x,y,Win.win,round(Nolap*Nfft),Nfft,fsmax);
      otherwise
        error('### Unknown method.')
    end

    % create new output fsdata
    fs = fsdata(f, txy, fsmax);
    fs = set(fs, 'name', sprintf('%s(%s->%s)', method, aj.data.name, ak.data.name));
    fs = set(fs, 'enbw', Win.nenbw*fsmax/Nfft);

    %---------- 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
      h = history(ALGONAME, VERSION, pl, [aj.hist ak.hist]);
    else
      h = history(ALGONAME, VERSION, pl, [ak.hist aj.hist]);
    end
    if na == length(invars)
      h = set(h, 'invars', [invars(j) invars(k)]);
      nk = invars{k};
      nj = invars{j};
    else
      nk = ak.name;
      nj = aj.name;
    end

    % make output analysis object
    b = ao(fs, h);

    % set name
    b = setnh(b, 'name', sprintf('%s(%s->%s)', method, nj, nk));

    % add to outputs
    bs(j,k) = {b};
  end % End second loop over AOs
end % End first loop over AOs

% now we have a cell matrix of AOs but
% we want a normal matrix
b  = [bs{:}];
varargout{1} = reshape(b, na, na);


%-----------------------------------------------------------------------
% 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


%-----------------------------------------------------------------------
% 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


```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

set_xy_axis

PURPOSE ^

SET_XY_AXIS set the x-axis and y-axis values.

SYNOPSIS ^

function data_out = set_xy_axis(data, x, y)

DESCRIPTION ^

```
 SET_XY_AXIS set the x-axis and y-axis values.

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

 DESCRIPTION: SET_XY_AXIS set the x-axis and y-axis values.

 CALL:       data = set_xy_axis (data,    x, y);
          ao.data = set_xy_axis (ao.data, x, y);

 VERSION: $Id: set_xy_axis.m,v 1.5 2008/02/12 22:11:21 hewitson Exp $

 HISTORY: 21-06-2007 Diepholz
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SOURCE CODE ^

```
function data_out = set_xy_axis(data, x, y)
% SET_XY_AXIS set the x-axis and y-axis values.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SET_XY_AXIS set the x-axis and y-axis values.
```

```
%
% CALL:       data = set_xy_axis (data,    x, y);
%          ao.data = set_xy_axis (ao.data, x, y);
%
% VERSION: $Id: set_xy_axis.m,v 1.5 2008/02/12 22:11:21 hewitson Exp $
%
% HISTORY: 21-06-2007 Diepholz
%             Creation
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: zero/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%   class: zero/private   %%%%%%%%%%%%%%%%%%

   <a href="matlab:help zero/private/fromxml">zero/private/fromxml</a>  -  read from
part of an XML tree and create a zero object.
   <a href="matlab:help zero/private/pfq2ri">zero/private/pfq2ri</a>   - Convert
frequency/Q pole/zero representation into real
   <a href="matlab:help zero/private/pri2fq">zero/private/pri2fq</a>   - Convert
comlpex pole/zero into frequency/Q pole/zero representation.
   <a href="matlab:help zero/private/retrieve">zero/private/retrieve</a> -  zero
objects with given ids from the LTPDA repository specified by the input
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get_xy_values

PURPOSE ^

GET_XY_VALUES returns the values in vals.

SYNOPSIS ^

function varargout = get_xy_values(data, pl)

DESCRIPTION ^

```
 GET_XY_VALUES returns the values in vals.

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

 DESCRIPTION: GET_XY_VALUES returns the values in vals.

 CALL:         y    = get_xy_values(ao.data);
              [x,y] = get_xy_values(ao.data);      --> x is an empty array
              [x,y] = get_xy_values(ao.data, pl);  --> the pl has no effect

 VERSION:      $Id: get_xy_values.m,v 1.5 2008/02/12 19:54:05 hewitson Exp $

 HISTORY:      25-05-2007 Diepholz
                  Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a cdata object into a string.

This function is called by:

SOURCE CODE ^

```
function varargout = get_xy_values(data, pl)
% GET_XY_VALUES returns the values in vals.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
% DESCRIPTION: GET_XY_VALUES returns the values in vals.
%
% CALL:         y    = get_xy_values(ao.data);
%              [x,y] = get_xy_values(ao.data);      --> x is an empty array
%              [x,y] = get_xy_values(ao.data, pl);  --> the pl has no effect
%
% VERSION:      $Id: get_xy_values.m,v 1.5 2008/02/12 19:54:05 hewitson Exp $
%
% HISTORY:      25-05-2007 Diepholz
%                  Creation
%
%%%
```

ltpda_xticks

PURPOSE ^

LTPDA_XTICKS set the input vector as the y-ticks of the current axis.

SYNOPSIS ^

function ltpda_xticks(v)

DESCRIPTION ^

```
 LTPDA_XTICKS set the input vector as the y-ticks of the current axis.

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

 DESCRIPTION: LTPDA_XTICKS set the input vector as the y-ticks of the current
              axis.

 CALL:        ltpda_xticks(v)

 VERSION:     $Id: ltpda_xticks.m,v 1.4 2007/07/16 12:52:20 ingo Exp $

 HISTORY:     10-08-2006 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_xticks(v)
% LTPDA_XTICKS set the input vector as the y-ticks of the current axis.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: LTPDA_XTICKS set the input vector as the y-ticks of the current
%              axis.
%
```

```
% CALL:        ltpda_xticks(v)
%
% VERSION:     $Id: ltpda_xticks.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
%
% HISTORY:     10-08-2006 M Hewitson
%                 Creation
%
%%%
```

set_xy_axis

PURPOSE ^

SET_XY_AXIS set the x-axis and y-axis values.

SYNOPSIS ^

function data_out = set_xy_axis(data, x, y)

DESCRIPTION ^

```
 SET_XY_AXIS set the x-axis and y-axis values.

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

 DESCRIPTION: SET_XY_AXIS set the x-axis and y-axis values.

 CALL:       data = set_xy_axis (data,    x, y);
          ao.data = set_xy_axis (ao.data, x, y);

 VERSION: $Id: set_xy_axis.m,v 1.5 2008/02/13 12:12:49 mauro Exp $

 HISTORY: 21-06-2007 Diepholz
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SOURCE CODE ^

```
function data_out = set_xy_axis(data, x, y)
% SET_XY_AXIS set the x-axis and y-axis values.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SET_XY_AXIS set the x-axis and y-axis values.
```

```
%
% CALL:       data = set_xy_axis (data,    x, y);
%          ao.data = set_xy_axis (ao.data, x, y);
%
% VERSION: $Id: set_xy_axis.m,v 1.5 2008/02/13 12:12:49 mauro Exp $
%
% HISTORY: 21-06-2007 Diepholz
%             Creation
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 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
overloads the foo function for Analysis Objects.
   <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
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get_xy_values

PURPOSE ^

GET_XY_VALUES returns the x-axis and/or y-axis values.

SYNOPSIS ^

function varargout = get_xy_values(data, pl)

DESCRIPTION ^

```
 GET_XY_VALUES returns the x-axis and/or y-axis values.

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

 DESCRIPTION: GET_XY_VALUES returns the x-axis and/or y-axis values.

 CALL:         x    = get_xy_values(ao.data);
             [x,y] = get_xy_values(ao.data);
             [x,y] = get_xy_values(ao.data, pl);
              y    = get_xy_values(ao.data, pl); --> only possible with
                                                     a parameter list

 REMARK:       In the case with TWO output arguments and only one parameter in
               'pl' the function returns a empty array for the axis which is
               not set in the parameterlist.

 PARAMETER LIST:  <key>   <value>   <description>
                 'xdata'    'x'     returns the values of the x-axis
                 'ydata'    'y'     returns the values of the y-axis

 VERSION:         $Id: get_xy_values.m,v 1.5 2008/02/12 22:11:21 hewitson Exp $

 HISTORY:         25-05-2007 Diepholz
                     Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SOURCE CODE ^

```
function varargout = get_xy_values(data, pl)
% GET_XY_VALUES returns the x-axis and/or y-axis values.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: GET_XY_VALUES returns the x-axis and/or y-axis values.
%
% CALL:         x    = get_xy_values(ao.data);
%             [x,y] = get_xy_values(ao.data);
%             [x,y] = get_xy_values(ao.data, pl);
%              y    = get_xy_values(ao.data, pl); --> only possible with
%                                                     a parameter list
%
% REMARK:       In the case with TWO output arguments and only one parameter in
%               'pl' the function returns a empty array for the axis which is
%               not set in the parameterlist.
%
% PARAMETER LIST:  <key>   <value>   <description>
%                 'xdata'    'x'     returns the values of the x-axis
%                 'ydata'    'y'     returns the values of the y-axis
%
% VERSION:         $Id: get_xy_values.m,v 1.5 2008/02/12 22:11:21 hewitson Exp $
%
% HISTORY:         25-05-2007 Diepholz
%                     Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: get_xy_values.m,v 1.5 2008/02/12 22:11:21 hewitson Exp $';
CATEGORY = 'Internal';

% Check if this is a call for parameters
if nargin == 2
  if isa(data, 'fsdata') && ischar(pl)
    in = char(pl);
    if strcmp(in, 'Params')
      varargout{1} = plist;
      return
    elseif strcmp(in, 'Version')
      varargout{1} = VERSION;
      return
    elseif strcmp(in, 'Category')
      varargout{1} = CATEGORY;
      return
    end
  end
end

if nargin < 2
```

```
  pl = [];
end

x  = [];
y  = [];

%% Set default parameter list
if nargout == 1
  pl_default = plist(param('ydata',  'y'));
else %if nargout == 2
  pl_default = plist([param('xdata',  'x')
                      param('ydata',  'y')]);
end


%% Combine the handover param-list and the default param-list
if ~isempty(pl)
  pl = combine(pl, pl_default);
else
  pl = pl_default;
end

do_xdata = find(pl, 'xdata');
do_ydata = find(pl, 'ydata');

%% Make sure that at least one axis is defined
if isempty(do_xdata) && isempty(do_ydata)
  do_xdata = 'x';
end

%% Set x-axis and y-axis values
if ~isempty(do_xdata)
  x = data.x;
end

if ~isempty(do_ydata)
  y = data.y;
end

%% Set outputs
if nargout == 1
  if ~isempty(y)
    varargout{1} = y;
  else % ~isempty(y)
    varargout{1} = x;
  end

elseif nargout == 2
    varargout{1} = x;
    varargout{2} = y;

else
  error('### Incorrect output arguments.');
```

```
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_yaxis

PURPOSE ^

LTPDA_YAXIS Set the Y axis range of the current figure

SYNOPSIS ^

function ltpda_yaxis(y1,y2)

DESCRIPTION ^

```
 LTPDA_YAXIS Set the Y axis range of the current figure

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

 DESCRIPTION: LTPDA_YAXIS Set the Y axis range of the current figure

 CALL:        ltpda_yaxis(x1,x2)

 VERSION:     $Id: ltpda_yaxis.m,v 1.4 2007/07/16 12:52:20 ingo Exp $

 HISTORY:     09-10-2001 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_yaxis(y1,y2)
% LTPDA_YAXIS Set the Y axis range of the current figure
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: LTPDA_YAXIS Set the Y axis range of the current figure
%
% CALL:        ltpda_yaxis(x1,x2)
%
```

```
% VERSION:     $Id: ltpda_yaxis.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
%
% HISTORY:     09-10-2001 M Hewitson
%                 Creation
%
%%%
```

set_xy_axis

PURPOSE ^

SET_XY_AXIS set the x-axis and y-axis values.

SYNOPSIS ^

function data_out = set_xy_axis(data, x, y)

DESCRIPTION ^

```
 SET_XY_AXIS set the x-axis and y-axis values.

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

 DESCRIPTION: SET_XY_AXIS set the x-axis and y-axis values.

 CALL:       data = set_xy_axis (data,    x, y);
          ao.data = set_xy_axis (ao.data, x, y);

 VERSION: $Id: set_xy_axis.m,v 1.4 2008/02/13 12:57:47 mauro Exp $

 HISTORY: 21-06-2007 Diepholz
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SOURCE CODE ^

```
function data_out = set_xy_axis(data, x, y)
% SET_XY_AXIS set the x-axis and y-axis values.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SET_XY_AXIS set the x-axis and y-axis values.
```

```
%
% CALL:       data = set_xy_axis (data,    x, y);
%          ao.data = set_xy_axis (ao.data, x, y);
%
% VERSION: $Id: set_xy_axis.m,v 1.4 2008/02/13 12:57:47 mauro Exp $
%
% HISTORY: 21-06-2007 Diepholz
%             Creation
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 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/explore_ao">m/gui/ao_browser/explore_ao</
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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get_xy_values

PURPOSE ^

GET_XY_VALUES returns the x-axis and/or y-axis values.

SYNOPSIS ^

function varargout = get_xy_values(data, pl)

DESCRIPTION ^

```
 GET_XY_VALUES returns the x-axis and/or y-axis values.

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

 DESCRIPTION: GET_XY_VALUES returns the x-axis and/or y-axis values.

 CALL:         x    = get_xy_values(ao.data);
             [x,y] = get_xy_values(ao.data);
             [x,y] = get_xy_values(ao.data, pl);
              y    = get_xy_values(ao.data, pl); --> only possible with
                                                     a parameter list

 REMARK:       In the case with TWO output arguments and only one parameter in
               'pl' the function returns a empty array for the axis which is
               not set in the parameterlist.

 PARAMETER LIST:  <key>   <value>   <description>
                 'xdata'    'x'     returns the values of the x-axis
                 'ydata'    'y'     returns the values of the y-axis

 VERSION:         $Id: get_xy_values.m,v 1.7 2008/02/13 11:39:35 mauro Exp $

 HISTORY:         25-05-2007 Diepholz
                     Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SOURCE CODE ^

```
function varargout = get_xy_values(data, pl)
% GET_XY_VALUES returns the x-axis and/or y-axis values.
%
%%%
```

```

if nargin < 2
  pl = [];
end

x  = [];
y  = [];

%% Set default parameter list
if nargout == 1
  pl_default = plist(param('ydata',  'y'));
else %if nargout == 2
  pl_default = plist([param('xdata',  'x')
                      param('ydata',  'y')]);
end


%% Combine the handover param-list and the default param-list
if ~isempty(pl)
  pl = combine(pl, pl_default);
else
  pl = pl_default;
end

do_xdata = find(pl, 'xdata');
do_ydata = find(pl, 'ydata');

%% Make sure that at least one axis is defined
if isempty(do_xdata) && isempty(do_ydata)
  do_xdata = 'x';
end

%% Set x-axis and y-axis values
if ~isempty(do_xdata)
  x = data.x;
end

if ~isempty(do_ydata)
  y = data.y;
end

%% Set outputs
if nargout == 1
  if ~isempty(y)
    varargout{1} = y;
  else % ~isempty(y)
    varargout{1} = x;
  end

elseif nargout == 2
    varargout{1} = x;
    varargout{2} = y;
```

```

else
  error('### Incorrect output arguments.');
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_yscale

PURPOSE ^

LTPDA_YSCALE Set the Y scale of the current axis

SYNOPSIS ^

function ltpda_yscale(scale)

DESCRIPTION ^

```
 LTPDA_YSCALE Set the Y scale of the current axis

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

 DESCRIPTION: LTPDA_YSCALE Set the Y scale of the current axis

 CALL:        ltpda_yscale('scale')         scale = 'lin' or 'log';

 VERSION:     $Id: ltpda_yscale.m,v 1.4 2007/07/16 12:52:20 ingo Exp $

 HISTORY:     02-05-2004 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_yscale(scale)
% LTPDA_YSCALE Set the Y scale of the current axis
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: LTPDA_YSCALE Set the Y scale of the current axis
%
% CALL:        ltpda_yscale('scale')         scale = 'lin' or 'log';
%
```

```
% VERSION:     $Id: ltpda_yscale.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
%
% HISTORY:     02-05-2004 M Hewitson
%                 Creation
%
%%%
```

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% path: m/gui/ltpdaRepoGUI %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   path: m/gui/ltpdaRepoGUI   %%%%%%%%%%%%%%%%%%%

   <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/uitabdemo">m/gui/ltpdaRepoGUI/uitabdemo</
a>     -   This script demostrates the use of UITABPANEL.
   <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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get_xy_values

PURPOSE ^

GET_XY_VALUES returns the x-axis and/or y-axis values.

SYNOPSIS ^

function varargout = get_xy_values(data, pl)

DESCRIPTION ^

```
 GET_XY_VALUES returns the x-axis and/or y-axis values.

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

 DESCRIPTION: GET_XY_VALUES returns the x-axis and/or y-axis values.

 CALL:         x    = get_xy_values(ao.data);
             [x,y] = get_xy_values(ao.data);
             [x,y] = get_xy_values(ao.data, pl);
              y    = get_xy_values(ao.data, pl); --> only possible with
                                                     a parameter list

 REMARK:       In the case with TWO output arguments and only one parameter in
               'pl' the function returns a empty array for the axis which is
               not set in the parameterlist.

 PARAMETER LIST:  <key>   <value>   <description>
                 'xdata'    'x'     returns the values of the x-axis
                 'ydata'    'y'     returns the values of the y-axis

 VERSION:         $Id: get_xy_values.m,v 1.4 2008/02/13 12:57:47 mauro Exp $

 HISTORY:         25-05-2007 Diepholz
                     Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SOURCE CODE ^

```
function varargout = get_xy_values(data, pl)
% GET_XY_VALUES returns the x-axis and/or y-axis values.
%
%%%
```

```

if nargin < 2
  pl = [];
end

x  = [];
y  = [];

%% Set default parameter list
if nargout == 1
  pl_default = plist(param('ydata',  'y'));
else %if nargout == 2
  pl_default = plist([param('xdata',  'x')
                      param('ydata',  'y')]);
end


%% Combine the handover param-list and the default param-list
if ~isempty(pl)
  pl = combine(pl, pl_default);
else
  pl = pl_default;
end

do_xdata = find(pl, 'xdata');
do_ydata = find(pl, 'ydata');

%% Make sure that at least one axis is defined
if isempty(do_xdata) && isempty(do_ydata)
  do_xdata = 'x';
end

%% Set x-axis and y-axis values
if ~isempty(do_xdata)
  x = data.x;
end

if ~isempty(do_ydata)
  y = data.y;
end

%% Set outputs
if nargout == 1
  if ~isempty(y)
    varargout{1} = y;
  else % ~isempty(y)
    varargout{1} = x;
  end

elseif nargout == 2
    varargout{1} = x;
    varargout{2} = y;
```

```
else
  error('### Incorrect output arguments.');
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_yticks

PURPOSE ^

LTPDA_YTICKS set the input vector as the y-ticks of the current axis.

SYNOPSIS ^

function ltpda_yticks(v)

DESCRIPTION ^

```
 LTPDA_YTICKS set the input vector as the y-ticks of the current axis.

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

 DESCRIPTION: LTPDA_YTICKS set the input vector as the y-ticks of the current
              axis.

 CALL:        ltpda_yticks(v)

 VERSION:     $Id: ltpda_yticks.m,v 1.4 2007/07/16 12:52:20 ingo Exp $

 HISTORY:     10-08-2006 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_yticks(v)
% LTPDA_YTICKS set the input vector as the y-ticks of the current axis.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: LTPDA_YTICKS set the input vector as the y-ticks of the current
%              axis.
%
```

```
% CALL:        ltpda_yticks(v)
%
% VERSION:     $Id: ltpda_yticks.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
%
% HISTORY:     10-08-2006 M Hewitson
%                 Creation
%
%%%
```

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>      - before deleting, record the table
selection
   <a href="matlab:help m/gui/ltpdaRepoGUI/callbacks/buildquery">m/gui/ltpdaRepoGUI/
callbacks/buildquery</a>           - Get number of conditions
   m/gui/ltpdaRepoGUI/callbacks/cb_condPlusBtn       - (No help available)
   m/gui/ltpdaRepoGUI/callbacks/cb_condSubtractBtn   - (No help available)
   <a href="matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_connectBtn">m/gui/
ltpdaRepoGUI/callbacks/cb_connectBtn</a>        - Get conn
   <a href="matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_dbSelect">m/gui/ltpdaRepoGUI/
callbacks/cb_dbSelect</a>          - Get selected DB
   <a href="matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_executeQuery">m/gui/
ltpdaRepoGUI/callbacks/cb_executeQuery</a>      - Get connection
   <a href="matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_fieldSelect">m/gui/
ltpdaRepoGUI/callbacks/cb_fieldSelect</a>       - Build query
   <a href="matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_getDBs">m/gui/ltpdaRepoGUI/
callbacks/cb_getDBs</a>            -  conn
   m/gui/ltpdaRepoGUI/callbacks/cb_mainfigClose      - (No help available)
   <a href="matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_refreshObjsList">m/gui/
ltpdaRepoGUI/callbacks/cb_refreshObjsList</a>   - get a list of objects
   <a href="matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_retrieveBtn">m/gui/
ltpdaRepoGUI/callbacks/cb_retrieveBtn</a>       - Get connection
   <a href="matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_submitBtn">m/gui/
ltpdaRepoGUI/callbacks/cb_submitBtn</a>         - get object list
   <a href="matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_tableSelect">m/gui/
ltpdaRepoGUI/callbacks/cb_tableSelect</a>       - Get conn
   <a href="matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_workspaceObjsList">m/gui/
ltpdaRepoGUI/callbacks/cb_workspaceObjsList</a> - get the selected line
   <a href="matlab:help m/gui/ltpdaRepoGUI/callbacks/getFields">m/gui/ltpdaRepoGUI/
callbacks/getFields</a>            -  the selected table
   <a href="matlab:help m/gui/ltpdaRepoGUI/callbacks/getTables">m/gui/ltpdaRepoGUI/
callbacks/getTables</a>            -  conn
```

```
   <a href="matlab:help m/gui/ltpdaRepoGUI/callbacks/getWorkspaceObjs">m/gui/
ltpdaRepoGUI/callbacks/getWorkspaceObjs</a>     -  base workspace variables
   <a href="matlab:help m/gui/ltpdaRepoGUI/callbacks/setWorkspaceObjsList">m/gui/
ltpdaRepoGUI/callbacks/setWorkspaceObjsList</a> - str = sprintf('%s\t\t(%s)', obj.
name, obj.class);
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

get_xyz_values

PURPOSE ^

GET_XYZ_VALUES returns the x-axis and/or y-axis and/or z-axis values.

SYNOPSIS ^

function varargout = get_xy_values(data, pl)

DESCRIPTION ^

```
 GET_XYZ_VALUES returns the x-axis and/or y-axis and/or z-axis values.

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

 DESCRIPTION: GET_XYZ_VALUES  returns the x-axis and/or y-axis and/or z-axis
                              values.

 CALL:          x    = get_xyz_values(ao.data);
              [x,y] = get_xyz_values(ao.data);
            [x,y,z] = get_xyz_values(ao.data);
              [x,y] = get_xyz_values(ao.data, pl);
            [x,y,z] = get_xyz_values(ao.data, pl);
               z    = get_xyz_values(ao.data, pl); --> only possible with
                                                      a parameter list

 REMARK:        In the case with 3 output arguments and only one parameter in
                'pl' the function returns a empty array for the axis which is
                not set in the parameterlist.

 PARAMETER LIST:  <key>   <value>   <description>
                 'xdata'    'x'     returns the values of the x-axis
                 'ydata'    'y'     returns the values of the y-axis
                 'zdata'    'z'     returns the values of the z-axis

 VERSION:         $Id: get_xyz_values.m,v 1.2 2008/02/13 13:22:45 mauro Exp $

 HISTORY:         24-12-2007 Hewitson
                     Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = get_xy_values(data, pl)
% GET_XYZ_VALUES returns the x-axis and/or y-axis and/or z-axis values.
%
%%%
```

```
    elseif strcmp(in, 'Category')
      varargout{1} = CATEGORY;
      return
    end
  end
end

if nargin < 2
  pl = [];
end

x  = [];
y  = [];

%% Set default parameter list
if nargout == 1
  pl_default = plist(param('xdata',  'x'));
elseif nargout == 2
  pl_default = plist([param('xdata',  'x')
                      param('ydata',  'y')]);
elseif nargout == 3
  pl_default = plist('xdata', 'x', 'ydata', 'y', 'zdata', 'z');
else
  error('### Incorrect outputs.');
end


%% Combine the handover param-list and the default param-list
if ~isempty(pl)
  pl = combine(pl);
else
  pl = combine(pl_default);
end

do_xdata = find(pl, 'xdata');
do_ydata = find(pl, 'ydata');
do_zdata = find(pl, 'zdata');

%% Make sure that at least one axis is defined
if isempty(do_xdata) && isempty(do_ydata)
  do_xdata = 'x';
end

%% Set x-axis and y-axis values
if ~isempty(do_xdata)
  x = data.x;
end

if ~isempty(do_ydata)
  y = data.y;
end

```

```
if ~isempty(do_zdata)
  z = data.z;
end

%% Set outputs
if nargout == 1
  if ~isempty(x)
    varargout{1} = x;
  else % ~isempty(y)
    varargout{1} = y;
  end

elseif nargout == 2
    varargout{1} = x;
    varargout{2} = y;

elseif nargout == 3
    varargout{1} = x;
    varargout{2} = y;
    varargout{3} = z;
else
  error('### Incorrect output arguments.');
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_zaxis

PURPOSE ^

LTPDA_ZAXIS Set the Z axis range of the current figure

SYNOPSIS ^

function ltpda_zaxis(x1,x2)

DESCRIPTION ^

```
 LTPDA_ZAXIS Set the Z axis range of the current figure

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

 DESCRIPTION: LTPDA_ZAXIS Set the Z axis range of the current figure

 CALL:        ltpda_zaxis(x1,x2)

 VERSION:     $Id: ltpda_zaxis.m,v 1.4 2007/07/16 12:52:20 ingo Exp $

 HISTORY:     09-10-2001 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_zaxis(x1,x2)
% LTPDA_ZAXIS Set the Z axis range of the current figure
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DESCRIPTION: LTPDA_ZAXIS Set the Z axis range of the current figure
%
% CALL:        ltpda_zaxis(x1,x2)
%
```

```
% VERSION:     $Id: ltpda_zaxis.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
%
% HISTORY:     09-10-2001 M Hewitson
%                 Creation
%
%%%
```

single_operation

PURPOSE ^

SINGLE_OPERATION implements specified operator overload for cdata objects.

SYNOPSIS ^

function varargout = single_operation (varargin)

DESCRIPTION ^

```
 SINGLE_OPERATION implements specified operator overload for cdata objects.

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

 DESCRIPTION: SINGLE_OPERATION implements specified operator overload
              for cdata objects.
              ATTENTION: Should only be called from a analysis function.

 CALL: [h, data_out] = single_operation(a.data, 'svd', pl)

 varargin format: 1:   [cdata]
                  2:   ['operation']
                  3:   [plist]

 varargout format 1: [new_history]
                  2: [data_out1]
                  3: [data_out2]      optional
                  4: [data_out3]      optional
                        ...           optional

 With the plist values you can control the command line.
 Possible key values are: option   <double> or <char>
                          dim      <double> or <char>
                          W        <vector>

 >> operation(data,'option')          option <char>
 >> operation(data,option)            option <double>
 >> operation(data,W,dim)
 >> operation(data,dim)
 >> operation(data,W)
 >> operation(data)

 HISTORY: 07-05-2007 Diepholz
             Creation.
```

```
%%%
```

```
%
% HISTORY: 07-05-2007 Diepholz
%             Creation.
%
%%%
```

```
  else
    pl = combine(pl_default);
  end

  option        = find(pl, 'option');
  dim           = find(pl, 'dim');
  weight_vector = find(pl, 'W');
  do_xdata      = find(pl, 'xdata');
  do_ydata      = find(pl, 'ydata');

  %% Check the <char> option if it begins and ends  with a quote '
  %% If not then add the quote.
  if ischar(option) && ~isempty(option)
    if ~strcmp(option(1), '''')
      option = ['''' option];
    end
    if ~strcmp(option(end), '''')
      option = [option ''''];
    end
  end

  %% Convert the <double> option into a char
  if isnumeric(option) || islogical(option)
    option = num2str(option);
  end

  %% Compute always the y data
  do_xdata = 'x';
  do_ydata = 'y';

  %% Convert the dim into a char
  if isnumeric(dim) || islogical (dim)
    dim = num2str(dim);
  end

  %% The operation can only compute with a 'option' or with a 'dimension'.
  %% e.g. operation(ao, option)
  %%      operation(ao, dim)
  if ~isempty(option) && ~isempty(dim)
    warning ('### xydata/single_operation: The param list contain a ''option''
and a ''dim''. It is computed %s(ao,option).', op);
  end

%% %% Ensure that the called function have at least two output variables
  if (nargout >= 2)

%%  %% Create the output string for the eval command
    %% e.g.: y_new = '[ y_new1 y_new2 ]'
    y_new = '[';
    for i = 1:nargout-1
      new_var_name = genvarname (sprintf('y_new%d',i));
      eval ([new_var_name '= 0;']);
```

```
      y_new = strcat(y_new, sprintf (' y_new%d',i));
    end
    y_new = strcat (y_new, ' ]');

%%  %% Create the output variables
    %% e.g.: data_out1 = data;
    for i = 1:nargout-1
      new_var_name = genvarname (sprintf('data_out%d',i));
      eval ([new_var_name '= data;']);
    end

%%  Execute the command

    try

      %% Execute with a option or a dimension
      %% >> operation(y,'option');
      if ~isempty(option)
        cmd_line = sprintf('%s = %s(y,%s);',y_new, op, option);
        eval (cmd_line);

      %% Execute with a weight vector and dimension
      %% >> operation(y,W,dim);
      elseif ~isempty(dim) && ~isempty(weight_vector)
        cmd_line = sprintf('%s = %s(y,weight_vector,%s);',y_new, op, dim);
        eval (cmd_line);

      %% Execute with a option or a dimension
      %% >> operation(y,dim);
      elseif ~isempty(dim)
        cmd_line = sprintf('%s = %s(y,%s);',y_new, op, dim);
        eval (cmd_line);

      %% Execute with a weight vector
      %% >> operation(y,W);
      elseif ~isempty(weight_vector)
        cmd_line = sprintf('%s = %s(y,weight_vector);',y_new, op);
        eval (cmd_line);

      %% Execute without parameter or option
      %% >> operation(y)
      else
        cmd_line = sprintf('%s = %s(y);',y_new, op);
        eval (cmd_line);
      end

    catch
      l_error = lasterror;
      disp(sprintf('\n\n%s', l_error.message));
      error('### cdata/single_operation: Can not command:\n###    %s\n### Have a look to the param list', cmd_line);
```

```
    end


%%  %% The result from eval is in y_new1, ...
    %% Create a cdata object from this results and write it into the output
    for i = 1:nargout-1

      %% Create cdata
      eval (sprintf('data_out%d.y = y_new%d;',i,i));
      eval (sprintf('data_out%d.name = ''%s(%s)'';',i,op,data.name));

      %% Write to output
      eval (sprintf('varargout{i+1} = data_out%d;',i))

    end

    %% Create a new history object
    h = history(op, VERSION);

    varargout{1} = h;

  else % (nargout < 2)
    error ('### cdata/single_operation: This function need at least two output variables.');
  end

else
  error ('### cdata/single_operation: This function need three input variables.')
end


function pl_default = getDefaultPL()

  pl_default = plist('option', '',  ...
                     'dim',    '',  ...
                     'W',      [],  ...
                     'xdata',  'x', ...
                     'ydata',  'y');

```

Generated on Tue 26-Feb-2008 10:52:52 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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

getlowerFreq

PURPOSE ^

GETLOWERFREQ gets the frequency of the lowest pole or zero in the model.

SYNOPSIS ^

function f = getlowerFreq(pzm, 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);

 VERSION:     $Id: getlowerFreq.m,v 1.4 2008/02/12 20:07:37 hewitson Exp $

 HISTORY:     04-04-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pzmodel object into a string.
- get GET get a pole/zero model property.

This function is called by:

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

SOURCE CODE ^

```
function f = getlowerFreq(pzm, varargin)
% GETLOWERFREQ gets the frequency of the lowest pole or zero in the model.
%
```

```
%%%
```

```
  end
end


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_zscale

PURPOSE ^

LTPDA_ZSCALE Set the Z scale of the current axis

SYNOPSIS ^

function ltpda_zscale(scale)

DESCRIPTION ^

```
 LTPDA_ZSCALE Set the Z scale of the current axis

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

 DESCRIPTION: LTPDA_ZSCALE Set the Z scale of the current axis

 CALL:        ltpda_zscale('scale')   scale = 'lin' or 'log';

 VERSION:     $Id: ltpda_zscale.m,v 1.4 2007/07/16 12:52:20 ingo Exp $

 HISTORY:     02-05-2004 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_zscale(scale)
% LTPDA_ZSCALE Set the Z scale of the current axis
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DESCRIPTION: LTPDA_ZSCALE Set the Z scale of the current axis
%
% CALL:        ltpda_zscale('scale')   scale = 'lin' or 'log';
%
```

```
% VERSION:     $Id: ltpda_zscale.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
%
% HISTORY:     02-05-2004 M Hewitson
%                 Creation
%
%%%
```

single_operation

PURPOSE ^

SINGLE_OPERATION implements specified operator overload for fsdata objects.

SYNOPSIS ^

function varargout = single_operation (varargin)

DESCRIPTION ^

```
 SINGLE_OPERATION implements specified operator overload for fsdata objects.

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

 DESCRIPTION: SINGLE_OPERATION implements specified operator overload
              for fsdata objects.
              ATTENTION: Should only be called from a analysis function.

 CALL: [h, data_out] = single_operation(a.data, 'log', pl)

 varargin format: 1:   [fsdata]
                  2:   ['operation']
                  3:   [plist]

 varargout format 1: [new_history]
                  2: [data_out1]
                  3: [data_out2]      optional
                  4: [data_out3]      optional
                        ...           optional

 With the plist values you can control the command line.
 Possible key values are: option   <double> or <char>
                          dim      <double> or <char>
                          W        <vector>
                          xdata    <char>
                          ydata    <char>
 With the keys xdata and ydata you can control which data are to compute.
 Attention the value of the key 'xdata' and 'ydata' must apply to the data class.
 The other values are controle the command line:

 >> operation(data,'option')          option <char>
 >> operation(data,option)            option <double>
 >> operation(data,W,dim)
 >> operation(data,dim)
 >> operation(data,W)
```

```
 >> operation(data)

 HISTORY: 07-05-2007 Diepholz
            Creation.

%%%
```

```
class.
% The other values are controle the command line:
%
% >> operation(data,'option')          option <char>
% >> operation(data,option)            option <double>
% >> operation(data,W,dim)
% >> operation(data,dim)
% >> operation(data,W)
% >> operation(data)
%
% HISTORY: 07-05-2007 Diepholz
%             Creation.
%
%%%
```

```
    error ('### fsdata/single_operation: The third input must be a plist.');
  end

  %% Set default parameter list
  pl_default = getDefaultPL();

  %% Combine the handover param-list and the default param-list
  if ~isempty(pl)
    pl = combine(pl);
  else
    pl = combine(pl_default);
  end

  option        = find(pl, 'option');
  dim           = find(pl, 'dim');
  weight_vector = find(pl, 'W');
  do_xdata      = find(pl, 'xdata');
  do_ydata      = find(pl, 'ydata');

  %% Check the <char> option if it begins and ends  with a quote '
  %% If not then add the quote.
  if ischar(option) && ~isempty(option)
    if ~strcmp(option(1), '''')
      option = ['''' option];
    end
    if ~strcmp(option(end), '''')
      option = [option ''''];
    end
  end

  %% Convert the <double> option into a char
  if isnumeric(option) || islogical(option)
    option = num2str(option);
  end

  %% Convert the dim into a char
  if isnumeric(dim) || islogical (dim)
    dim = num2str(dim);
  end

  %% The operation can only compute with a 'option' or with a 'dimension'.
  %% e.g. operation(ao, 'option')
  %%      operation(ao, dim)
  if ~isempty(option) && ~isempty(dim)
    warning ('### xydata/single_operation: The param list contain a ''option''
and a ''dim''. It is computed %s(ao,option).', op);
  end

  %%
  if isempty(do_xdata) && isempty(do_ydata)
    do_ydata = 'y';
  end

```

```
%% %% Ensure that the called function have at least two output variables
  if (nargout >= 2)

%%  %% y (y-axis):
    %% Create the output string for the eval command
    %% e.g.: y_new = '[ y_new1 y_new2 ]'
    y_new = '[';
    for i = 1:nargout-1
      new_var_name = genvarname (sprintf('y_new%d',i));
      eval ([new_var_name '= 0;']);

      y_new = strcat(y_new, sprintf (' y_new%d',i));
    end
    y_new = strcat (y_new, ' ]');

%%  %% t (x-axis):
    %% Create the output string for the eval command
    %% e.g.: x_new = '[ x_new1 x_new2 ]'
    x_new = '[';
    for i = 1:nargout-1
      new_var_name = genvarname (sprintf('x_new%d',i));
      eval ([new_var_name '= 0;']);

      x_new = strcat(x_new, sprintf (' x_new%d',i));
    end
    x_new = strcat (x_new, ' ]');

%%  %% Create the output variables
    %% e.g.: data_out1 = data;
    for i = 1:nargout-1
      new_var_name = genvarname (sprintf('data_out%d',i));
      eval ([new_var_name '= data;']);
    end

%%  Execute the command for y (y-axis)
    if ~isempty(do_ydata)

      try
        %% Execute with a option
        %% >> operation(y,'option');
        if ~isempty(option)
          cmd_line = sprintf('%s = %s(y,%s);',y_new, op, option);
          eval (cmd_line);

        %% Execute with a weight vector and dimension
        %% >> operation(y,W,dim);
        elseif ~isempty(dim) && ~isempty(weight_vector)
          cmd_line = sprintf('%s = %s(y,weight_vector,%s);',y_new, op, dim);
          eval (cmd_line);

        %% Execute with a dimension
        %% >> operation(y,dim);
        elseif ~isempty(dim)
```

```
          cmd_line = sprintf('%s = %s(y,%s);',y_new, op, dim);
          eval (cmd_line);

        %% Execute with a weight vector
        %% >> operation(y,W);
        elseif ~isempty(weight_vector)
          cmd_line = sprintf('%s = %s(y,weight_vector);',y_new, op);
          eval (cmd_line);

        %% Execute without parameter or option
        %% >> operation(y)
        else
          cmd_line = sprintf('%s = %s(y);',y_new, op);
          eval (cmd_line);
        end

      catch
        l_error = lasterror;
        disp(sprintf('\n\n%s', l_error.message));
        error('### cdata/single_operation: Can not command:\n###    %s\n### Have a look to the param list', cmd_line);
      end

    end

%%  Execute the command for x (x-axis)
    if ~isempty(do_xdata)

      try
        %% Execute with a option or a dimension
        %% >> operation(x,'option');
        if ~isempty(option)
          cmd_line = sprintf('%s = %s(x,%s);',x_new, op, option);
          eval (cmd_line);

        %% Execute with a weight vector and dimension
        %% >> operation(x,W,dim);
        elseif ~isempty(dim) && ~isempty(weight_vector)
          cmd_line = sprintf('%s = %s(x,weight_vector,%s);',x_new, op, dim);
          eval (cmd_line);

        %% Execute with a option or a dimension
        %% >> operation(x,dim);
        elseif ~isempty(dim)
          cmd_line = sprintf('%s = %s(x,%s);',x_new, op, dim);
          eval (cmd_line);

        %% Execute with a weight vector
        %% >> operation(x,W);
        elseif ~isempty(weight_vector)
          cmd_line = sprintf('%s = %s(x,weight_vector);',x_new, op);
          eval (cmd_line);

```

```
        %% Execute without parameter or option
        %% >> operation(x)
        else
          cmd_line = sprintf('%s = %s(x);',x_new, op);
          eval (cmd_line);
        end

      catch
        l_error = lasterror;
        disp(sprintf('\n\n%s', l_error.message));
        error('### cdata/single_operation: Can not command:\n###    %s\n### Have a look to the param list', cmd_line);
      end

    end

%%  %% The result from eval is in y_new1, x_new1, ...
    %% Create a fsdata object from this results and write it into the output
    for i = 1:nargout-1

      %% Create fsdata
      %% y data (y-axis)
      if ~isempty(do_ydata)
        eval (sprintf('data_out%d.y = y_new%d;',i,i));
        eval (sprintf('data_out%d.name = ''%s(%s)'';',i,op,data.name));
        eval (sprintf('data_out%d.yunits = ''%s(%s)'';',i,op,data.yunits));
      end
      %% t data (x-axis)
      if ~isempty(do_xdata)
        eval (sprintf('data_out%d.x = x_new%d;',i,i));
        eval (sprintf('data_out%d.name = ''%s(%s)'';',i,op,data.name));
        eval (sprintf('data_out%d.xunits = ''%s(%s)'';',i,op,data.xunits));
      end

      %% Write to output
      eval (sprintf('varargout{i+1} = data_out%d;',i))

    end

    %% Create a new history object
    h = history(op, VERSION);

    varargout{1} = h;

  else % (nargout < 2)
    error ('### fsdata/single_operation: This function need at least two output variables.');
  end

else
  error ('### fsdata/single_operation: This function need three input variables.')
end
```

```


function pl_default = getDefaultPL()

  pl_default = plist('option', '',  ...
                     'dim',    '',  ...
                     'W',      [],  ...
                     'xdata',  '',  ...
                     'ydata',  'y');

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% path: m/helper %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   path: m/helper   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help m/helper/dunzip">m/helper/dunzip</a>                    -
- decompress DZIP output to recover original data
   <a href="matlab:help m/helper/dzip">m/helper/dzip</a>                       -  -
losslessly compress data into smaller memory space
   <a href="matlab:help m/helper/gcbsh">m/helper/gcbsh</a>                      -
gets the handles for the currently selected blocks.
   <a href="matlab:help m/helper/get_curr_m_file_path">m/helper/
get_curr_m_file_path</a>       -  returns the path for a mfile.
   <a href="matlab:help m/helper/ltpda_deg2rad">m/helper/ltpda_deg2rad</
a>              -  Convert degrees to radians
   <a href="matlab:help m/helper/ltpda_fields2list">m/helper/ltpda_fields2list</
a>          -  splits a string containing fields seperated by ','
   <a href="matlab:help m/helper/ltpda_filescan">m/helper/ltpda_filescan</
a>             -  recursively scans the given directory for files that end in 'ext'
   <a href="matlab:help m/helper/ltpda_fq2ri">m/helper/ltpda_fq2ri</
a>                -  Convert frequency/Q pole/zero representation into real
   <a href="matlab:help m/helper/ltpda_fq2ri2">m/helper/ltpda_fq2ri2</
a>               -  Convert frequency/Q pole/zero representation into real
   <a href="matlab:help m/helper/ltpda_getmax">m/helper/ltpda_getmax</
a>               -  get the index and value of the maximum value in an array.
   <a href="matlab:help m/helper/ltpda_hash">m/helper/ltpda_hash</a>
-  - Convert an input variable into a message digest using any of
   <a href="matlab:help m/helper/ltpda_io_collect">m/helper/ltpda_io_collect</
a>           -  collects all AOs and plists from the inputs.
   <a href="matlab:help m/helper/ltpda_isobject">m/helper/ltpda_isobject</
a>             -  checks that the input objects are one of the LTPDA object
   <a href="matlab:help m/helper/ltpda_label">m/helper/ltpda_label</
a>                -  makes the input string into a suitable string for using on
plots.
   <a href="matlab:help m/helper/ltpda_mat2str">m/helper/ltpda_mat2str</
a>              -  overloads the mat2str operator to set the precision at a central
place.
   <a href="matlab:help m/helper/ltpda_matchvectors">m/helper/ltpda_matchvectors</
```

```
a>         - LTPDA_RESAMPLE make the two vectors have the same length.
   <a href="matlab:help m/helper/ltpda_num2str">m/helper/ltpda_num2str</
a>              -  uses sprintf to convert a data vector to a string with a fixed
precision.
   <a href="matlab:help m/helper/ltpda_phase">m/helper/ltpda_phase</
a>                -  return the phase in degrees for a given complex input.
   <a href="matlab:help m/helper/ltpda_rand">m/helper/ltpda_rand</a>
-  return a random number between r1 and r2
   <a href="matlab:help m/helper/ltpda_ri2fq">m/helper/ltpda_ri2fq</
a>                -  Convert complex pole/zero into frequency/Q pole/zero
representation.
   <a href="matlab:help m/helper/ltpda_saveobj">m/helper/ltpda_saveobj</
a>              -  saves an object to a file.
   <a href="matlab:help m/helper/ltpda_str2cells">m/helper/ltpda_str2cells</
a>            -  Take a single string and separate out individual "elements" into a
new cell array.
   <a href="matlab:help m/helper/ltpda_strs2cells">m/helper/ltpda_strs2cells</
a>           -  convert a set of input strings to a cell array.
   <a href="matlab:help m/helper/ltpda_structcat">m/helper/ltpda_structcat</
a>            -  concatonate structures to make one large structure.
   <a href="matlab:help m/helper/ltpda_structidx">m/helper/ltpda_structidx</
a>            -  Index a structure.
   <a href="matlab:help m/helper/ltpda_unwrapdeg">m/helper/ltpda_unwrapdeg</
a>            -  Unwrap a phase vector given in degrees.
   <a href="matlab:help m/helper/ltpda_xml_read_datasamples">m/helper/
ltpda_xml_read_datasamples</a> -  read data samples from the given node of an XML
file.
   <a href="matlab:help m/helper/rstruct">m/helper/rstruct</a>                   -
recursively converts an object into a structure. This is the same
   <a href="matlab:help m/helper/strpad">m/helper/strpad</a>                    -
Pads a string with blank spaces until it is N characters long.
   <a href="matlab:help m/helper/wrapstring">m/helper/wrapstring</a>
-  wraps a string to a cell array of strings with each cell less than n characters
long.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

getupperFreq

PURPOSE ^

GETUPPERFREQ gets the frequency of the highest pole or zero in the model.

SYNOPSIS ^

function f = getupperFreq(pzm, varargin)

DESCRIPTION ^

```
 GETUPPERFREQ gets the frequency of the highest pole or zero in the model.

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

 DESCRIPTION: GETUPPERFREQ gets the frequency of the highest pole or zero in
              the model.

 CALL:        f = getupperFreq(pzm);

 VERSION:     $Id: getupperFreq.m,v 1.4 2008/02/12 20:07:37 hewitson Exp $

 HISTORY:     04-04-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pzmodel object into a string.
- get GET get a pole/zero model property.

This function is called by:

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

SOURCE CODE ^

```
function f = getupperFreq(pzm, varargin)
% GETUPPERFREQ gets the frequency of the highest pole or zero in the model.
%
```

```
%%%
```

```
  end
end


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

single_operation

PURPOSE ^

SINGLE_OPERATION implements specified operator overload for tsdata objects.

SYNOPSIS ^

function varargout = single_operation (varargin)

DESCRIPTION ^

```
 SINGLE_OPERATION implements specified operator overload for tsdata objects.

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

 DESCRIPTION: SINGLE_OPERATION implements specified operator overload
              for tsdata objects.
              ATTENTION: Should only be called from a analysis function.

 CALL: [h, data_out] = single_operation(a.data, 'log', pl)

 varargin format: 1:   [tsdata]
                  2:   ['operation']
                  3:   [plist]

 varargout format 1: [new_history]
                  2: [data_out1]
                  3: [data_out2]      optional
                  4: [data_out3]      optional
                        ...           optional

 With the plist values you can control the command line.
 Possible key values are: option   <double> or <char>
                          dim      <double> or <char>
                          W        <vector>
                          xdata    <char>
                          ydata    <char>
 With the keys xdata and ydata you can control which data are to compute.
 Attention the value of the key 'xdata' and 'ydata' must apply to the data class.
 The other values are controle the command line:

 >> operation(data,'option')          option <char>
 >> operation(data,option)            option <double>
 >> operation(data,W,dim)
 >> operation(data,dim)
 >> operation(data,W)
```

```
 >> operation(data)

 HISTORY: 07-05-2007 Diepholz
            Creation.

%%%
```

```
class.
% The other values are controle the command line:
%
% >> operation(data,'option')          option <char>
% >> operation(data,option)            option <double>
% >> operation(data,W,dim)
% >> operation(data,dim)
% >> operation(data,W)
% >> operation(data)
%
% HISTORY: 07-05-2007 Diepholz
%             Creation.
%
%%%
```

```
    error ('### tsdata/single_operation: The third input must be a plist.');
  end

  %% Set default parameter list
  pl_default = getDefaultPL();

  %% Combine the handover param-list and the default param-list
  if ~isempty(pl)
    pl = combine(pl);
  else
    pl = combine(pl_default);
  end

  option        = find(pl, 'option');
  dim           = find(pl, 'dim');
  weight_vector = find(pl, 'W');
  do_xdata      = find(pl, 'xdata');
  do_ydata      = find(pl, 'ydata');

  %% Check the <char> option if it begins and ends  with a quote '
  %% If not then add the quote.
  if ischar(option) && ~isempty(option)
    if ~strcmp(option(1), '''')
      option = ['''' option];
    end
    if ~strcmp(option(end), '''')
      option = [option ''''];
    end
  end

  %% Convert the <double> option into a char
  if isnumeric(option) || islogical(option)
    option = num2str(option);
  end

  %% Convert the dim into a char
  if isnumeric(dim) || islogical(dim)
    dim = num2str(dim);
  end

  %% The operation can only compute with a 'option' or with a 'dimension'.
  %% e.g. operation(ao, 'option')
  %%      operation(ao, dim)
  if ~isempty(option) && ~isempty(dim)
    warning ('### xydata/single_operation: The param list contain a ''option''
and a ''dim''. It is computed %s(ao,option).', op);
  end

  %%
  if isempty(do_xdata) && isempty(do_ydata)
    do_ydata = 'y';
  end

```

```
%% %% Ensure that the called function have at least two output variables
  if (nargout >= 2)

%%  %% y (y-axis):
    %% Create the output string for the eval command
    %% e.g.: y_new = '[ y_new1 y_new2 ]'
    y_new = '[';
    for i = 1:nargout-1
      new_var_name = genvarname (sprintf('y_new%d',i));
      eval ([new_var_name '= 0;']);

      y_new = strcat(y_new, sprintf (' y_new%d',i));
    end
    y_new = strcat (y_new, ' ]');

%%  %% x (x-axis):
    %% Create the output string for the eval command
    %% e.g.: x_new = '[ x_new1 x_new2 ]'
    x_new = '[';
    for i = 1:nargout-1
      new_var_name = genvarname (sprintf('x_new%d',i));
      eval ([new_var_name '= 0;']);

      x_new = strcat(x_new, sprintf (' x_new%d',i));
    end
    x_new = strcat (x_new, ' ]');

%%  %% Create the output variables
    %% e.g.: data_out1 = data;
    for i = 1:nargout-1
      new_var_name = genvarname (sprintf('data_out%d',i));
      eval ([new_var_name '= data;']);
    end

%%  Execute the command for y (y-axis)
    if ~isempty(do_ydata)

      try
        %% Execute with a option
        %% >> operation(y,'option');
        if ~isempty(option)
          cmd_line = sprintf('%s = %s(y,%s);',y_new, op, option);
          eval (cmd_line);

        %% Execute with a weight vector and dimension
        %% >> operation(y,W,dim);
        elseif ~isempty(dim) && ~isempty(weight_vector)
          cmd_line = sprintf('%s = %s(y,weight_vector,%s);',y_new, op, dim);
          eval (cmd_line);

        %% Execute with a dimension
        %% >> operation(y,dim);
        elseif ~isempty(dim)
```

```
          cmd_line = sprintf('%s = %s(y,%s);',y_new, op, dim);
          eval (cmd_line);

        %% Execute with a weight vector
        %% >> operation(y,W);
        elseif ~isempty(weight_vector)
          cmd_line = sprintf('%s = %s(y,weight_vector);',y_new, op);
          eval (cmd_line);

        %% Execute without parameter or option
        %% >> operation(y)
        else
          cmd_line = sprintf('%s = %s(y);',y_new, op);
          eval (cmd_line);
        end

      catch
        l_error = lasterror;
        disp(sprintf('\n\n%s', l_error.message));
        error('### cdata/single_operation: Can not command:\n###    %s\n### Have a look to the param list', cmd_line);
      end

    end

%%  Execute the command for x (x-axis)
    if ~isempty(do_xdata)

      try
        %% Execute with a option or a dimension
        %% >> operation(x,'option');
        if ~isempty(option)
          cmd_line = sprintf('%s = %s(x,%s);',x_new, op, option);
          eval (cmd_line);

        %% Execute with a weight vector and dimension
        %% >> operation(x,W,dim);
        elseif ~isempty(dim) && ~isempty(weight_vector)
          cmd_line = sprintf('%s = %s(x,weight_vector,%s);',x_new, op, dim);
          eval (cmd_line);

        %% Execute with a option or a dimension
        %% >> operation(x,dim);
        elseif ~isempty(dim)
          cmd_line = sprintf('%s = %s(x,%s);',x_new, op, dim);
          eval (cmd_line);

        %% Execute with a weight vector
        %% >> operation(x,W);
        elseif ~isempty(weight_vector)
          cmd_line = sprintf('%s = %s(x,weight_vector);',x_new, op);
          eval (cmd_line);

```

```
        %% Execute without parameter or option
        %% >> operation(x)
        else
          cmd_line = sprintf('%s = %s(x);',x_new, op);
          eval (cmd_line);
        end

      catch
        l_error = lasterror;
        disp(sprintf('\n\n%s', l_error.message));
        error('### cdata/single_operation: Can not command:\n###    %s\n### Have a look to the param list', cmd_line);
      end

    end

%%  %% The result from eval is in y_new1, x_new1, ...
    %% Create a tsdata object from this results and write it into the output
    for i = 1:nargout-1

      %% Create tsdata
      %% y data (y-axis)
      if ~isempty(do_ydata)
        eval (sprintf('data_out%d.y = y_new%d;',i,i));
        eval (sprintf('data_out%d.name = ''%s(%s)'';',i,op,data.name));
        eval (sprintf('data_out%d.yunits = ''%s(%s)'';',i,op,data.yunits));
      end
      %% x data (x-axis)
      if ~isempty(do_xdata)
        eval (sprintf('data_out%d.x = x_new%d;',i,i));
        eval (sprintf('data_out%d.name = ''%s(%s)'';',i,op,data.name));
        eval (sprintf('data_out%d.xunits = ''%s(%s)'';',i,op,data.xunits));
      end

      %% Write to output
      eval (sprintf('varargout{i+1} = data_out%d;',i))

    end

    %% Create a new history object
    h = history(op, VERSION);

    varargout{1} = h;

  else % (nargout < 2)
    error ('### tsdata/single_operation: This function need at least two output variables.');
  end

else
  error ('### tsdata/single_operation: This function need three input variables.')
end
```

```

function pl_default = getDefaultPL()

  pl_default = plist('option', '',  ...
                     'dim',    '',  ...
                     'W',      [],  ...
                     'xdata',  '',  ...
                     'ydata',  'y');

```

Generated on Tue 26-Feb-2008 10:52:52 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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

single_operation

PURPOSE ^

SINGLE_OPERATION implements specified operator overload for xydata objects.

SYNOPSIS ^

function varargout = single_operation (varargin)

DESCRIPTION ^

```
 SINGLE_OPERATION implements specified operator overload for xydata objects.

%%%
```

```
 >> operation(data)

 HISTORY: 07-05-2007 Diepholz
            Creation.

%%%
```

```
class.
% The other values are controle the command line:
%
% >> operation(data,'option')         option <char>
% >> operation(data,option)           option <double>
% >> operation(data,W,dim)
% >> operation(data,dim)
% >> operation(data,W)
% >> operation(data)
%
% HISTORY: 07-05-2007 Diepholz
%             Creation.
%
%%%
```

```
    error ('### xydata/single_operation: The third input must be a plist.');
  end

  %% Set default parameter list
  pl_default = getDefaultPL();

  %% Combine the handover param-list and the default param-list
  if ~isempty(pl)
    pl = combine(pl);
  else
    pl = combine(pl_default);
  end

  option        = find(pl, 'option');
  dim           = find(pl, 'dim');
  weight_vector = find(pl, 'W');
  do_xdata      = find(pl, 'xdata');
  do_ydata      = find(pl, 'ydata');

  %% Check the <char> option if it begins and ends  with a quote '
  %% If not then add the quote.
  if ischar(option) && ~isempty(option)
    if ~strcmp(option(1), '''')
      option = ['''' option];
    end
    if ~strcmp(option(end), '''')
      option = [option ''''];
    end
  end

  %% Convert the <double> option into a char
  if isnumeric(option) || islogical(option)
    option = num2str(option);
  end

  %% Convert the dim into a char
  if isnumeric(dim) || islogical(dim)
    dim = num2str(dim);
  end

  %% The operation can only compute with a 'option' or with a 'dimension'.
  %% e.g. operation(ao, 'option')
  %%      operation(ao, dim)
  if ~isempty(option) && ~isempty(dim)
    warning ('### xydata/single_operation: The param list contain a ''option'' and a ''dim''. It is computed %s(ao,option).', op);
  end

  %%
  if isempty(do_xdata) && isempty(do_ydata)
    do_ydata = 'y';
  end

```

```
%% %% Ensure that the called function have at least two output variables
  if (nargout >= 2)

%%  %% xdata:
    %% Create the output string for the eval command
    %% e.g.: x_new = '[ x_new1 x_new2 ]'
    x_new = '[';
    for i = 1:nargout-1
      new_var_name = genvarname (sprintf('x_new%d',i));
      eval ([new_var_name '= 0;']);

      x_new = strcat(x_new, sprintf (' x_new%d',i));
    end
    x_new = strcat (x_new, ' ]');

    %% ydata:
    %% Create the output string for the eval command
    %% e.g.: y_new = '[ y_new1 y_new2 ]'
    y_new = '[';
    for i = 1:nargout-1
      new_var_name = genvarname (sprintf('y_new%d',i));
      eval ([new_var_name '= 0;']);

      y_new = strcat(y_new, sprintf (' y_new%d',i));
    end
    y_new = strcat (y_new, ' ]');

%%  %% Create the output variables
    %% e.g.: data_out1 = data;
    for i = 1:nargout-1
      new_var_name = genvarname (sprintf('data_out%d',i));
      eval ([new_var_name '= data;']);
    end

%%  Execute the command for the xdata
    if ~isempty(do_xdata)

      try
        %% Execute with a option
        %% >> operation(x,'option');
        if ~isempty(option)
          cmd_line = sprintf('%s = %s(x,%s);',x_new, op, option);
          eval (cmd_line);

        %% Execute with a weight vector and dimension
        %% >> operation(x,W,dim);
        elseif ~isempty(dim) && ~isempty(weight_vector)
          cmd_line = sprintf('%s = %s(x,weight_vector,%s);',x_new, op, dim);
          eval (cmd_line);

        %% Execute with a dimension
        %% >> operation(x,dim);
        elseif ~isempty(dim)
```

```
          cmd_line = sprintf('%s = %s(x,%s);',x_new, op, dim);
          eval (cmd_line);

        %% Execute with a weight vector
        %% >> operation(x,W);
        elseif ~isempty(weight_vector)
          cmd_line = sprintf('%s = %s(x,weight_vector);',x_new, op);
          eval (cmd_line);

        %% Execute without parameter or option
        %% >> operation(x)
        else
          cmd_line = sprintf('%s = %s(x);',x_new, op);
          eval (cmd_line);
        end

      catch
        l_error = lasterror;
        disp(sprintf('\n\n%s', l_error.message));
        error('### cdata/single_operation: Can not command:\n###    %s\n### Have a look to the param list', cmd_line);
      end

    end

%%  Execute the command for the ydata
    if ~isempty(do_ydata)

      try
        %% Execute with a option or a dimension
        %% >> operation(y,'option');
        if ~isempty(option)
          cmd_line = sprintf('%s = %s(y,%s);',y_new, op, option);
          eval (cmd_line);

        %% Execute with a weight vector and dimension
        %% >> operation(y,W,dim);
        elseif ~isempty(dim) && ~isempty(weight_vector)
          cmd_line = sprintf('%s = %s(y,weight_vector,%s);',y_new, op, dim);
          eval (cmd_line);

        %% Execute with a option or a dimension
        %% >> operation(y,dim);
        elseif ~isempty(dim)
          cmd_line = sprintf('%s = %s(y,%s);',y_new, op, dim);
          eval (cmd_line);

        %% Execute with a weight vector
        %% >> operation(y,W);
        elseif ~isempty(weight_vector)
          cmd_line = sprintf('%s = %s(y,weight_vector);',y_new, op);
          eval (cmd_line);

```

```
        %% Execute without parameter or option
        %% >> operation(y)
        else
          cmd_line = sprintf('%s = %s(y);',y_new, op);
          eval (cmd_line);
        end

      catch
        l_error = lasterror;
        disp(sprintf('\n\n%s', l_error.message));
        error('### cdata/single_operation: Can not command:\n###    %s\n### Have a look to the param list', cmd_line);
      end

    end

%%  %% The result from eval is in x_new1, y_new1, ...
    %% Create a xydata object from this results and write it into the output
    for i = 1:nargout-1

      %% Create xydata
      if ~isempty(do_xdata)
        eval (sprintf('data_out%d.x = x_new%d;',i,i));
        eval (sprintf('data_out%d.name = ''%s(%s)'';',i,op,data.name));
        eval (sprintf('data_out%d.xunits = ''%s(%s)'';',i,op,data.xunits));
      end
      if ~isempty(do_ydata)
        eval (sprintf('data_out%d.y = y_new%d;',i,i));
        eval (sprintf('data_out%d.name = ''%s(%s)'';',i,op,data.name));
        eval (sprintf('data_out%d.yunits = ''%s(%s)'';',i,op,data.yunits));
      end

      %% Write to output
      eval (sprintf('varargout{i+1} = data_out%d;',i))

    end

    %% Create a new history object
    h = history(op, VERSION);

    varargout{1} = h;

  else % (nargout < 2)
    error ('### xydata/single_operation: This function need at least two output variables.');
  end

else
  error ('### xydata/single_operation: This function need three input variables.')
end

function pl_default = getDefaultPL()
```

```

  pl_default = plist('option', '',  ...
                     'dim',    '',  ...
                     'W',      [],  ...
                     'xdata',  '',  ...
                     'ydata',  'y');

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% path: m/mysql %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   path: m/mysql   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help m/mysql/ltpda_dbquery">m/mysql/ltpda_dbquery</a>      -
query an AO repository database.
   <a href="matlab:help m/mysql/ltpda_insert">m/mysql/ltpda_insert</a>       -
inserts values into a single row of a table using JDBC driver
   <a href="matlab:help m/mysql/ltpda_obj_retrieve">m/mysql/ltpda_obj_retrieve</a>
-  retrieves a collection of objects from an LTPDA repository.
   <a href="matlab:help m/mysql/ltpda_obj_submit">m/mysql/ltpda_obj_submit</a>   -
submits the given collection of objects to an LTPDA Repository.
   <a href="matlab:help m/mysql/mysql_connect">m/mysql/mysql_connect</a>      -
connects to an AO repository and returns the connection object.
   <a href="matlab:help m/mysql/mysql_getMD5hash">m/mysql/mysql_getMD5hash</a>   -
gets the md5 hash string of a given object ID.
   <a href="matlab:help m/mysql/mysql_getMaxId">m/mysql/mysql_getMaxId</a>     -
get the maximum Id from the objs table.
   <a href="matlab:help m/mysql/mysql_getObjIds">m/mysql/mysql_getObjIds</a>    -
gets a list of object ids from a given collection id.
   <a href="matlab:help m/mysql/mysql_getObjType">m/mysql/mysql_getObjType</a>   -
gets the object type associated with the given object id.
   <a href="matlab:help m/mysql/mysql_getUserID">m/mysql/mysql_getUserID</a>    -
gets the user ID number corresponding to the given user name.
   <a href="matlab:help m/mysql/mysql_getXdoc">m/mysql/mysql_getXdoc</a>      -
retrieves an object with given id from the LTPDA
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

---

single_operation

PURPOSE ^

SINGLE_OPERATION implements specified operator overload for xyzdata objects.

SYNOPSIS ^

function varargout = single_operation (varargin)

DESCRIPTION ^

```
 SINGLE_OPERATION implements specified operator overload for xyzdata objects.

%%%
```

```
 data class.
 The other values are to control the command line:

 >> operation(data,'option')         option <char>
 >> operation(data,option)           option <double>
 >> operation(data,W,dim)
 >> operation(data,dim)
 >> operation(data,W)
 >> operation(data)

 HISTORY: 24-12-2007 Diepholz
             Creation.

%%%
```

```
%                   3: [data_out2]      optional
%                   4: [data_out3]      optional
%                          ...          optional
%
% With the plist values you can control the command line.
% Possible key values are: option   <double> or <char>
%                          dim      <double> or <char>
%                          W        <vector>
%                          xdata    <char>
%                          ydata    <char>
%                          xdata    <char>
% With the keys xdata,ydata or zdata you can control which data are to compute.
% Attention the value of the key 'xdata', 'ydata' or 'zdata' must apply to the
% data class.
% The other values are to control the command line:
%
% >> operation(data,'option')         option <char>
% >> operation(data,option)           option <double>
% >> operation(data,W,dim)
% >> operation(data,dim)
% >> operation(data,W)
% >> operation(data)
%
% HISTORY: 24-12-2007 Diepholz
%             Creation.
%
%%%
```

```
  pl   = varargin {3};
  a    = varargin {end};
  x    = data.x;
  y    = data.y;
  z    = data.z;

%% Verify input variables and set default parameter
  if ~isa(data, 'xyzdata')
    error ('### xyzdata/single_operation: The first input must be a xyzdata.');
  end
  if ~ischar(op)
    error ('### xyzdata/single_operation: The second input must be a string.');
  end
  if ~isa(plist, 'plist')
    error ('### xyzdata/single_operation: The third input must be a plist.');
  end

  %% Set default parameter list
  pl_default = getDefaultPL();

  %% Combine the handover param-list and the default param-list
  if ~isempty(pl)
    pl = combine(pl);
  else
    pl = combine(pl_default);
  end

  option        = find(pl, 'option');
  dim           = find(pl, 'dim');
  weight_vector = find(pl, 'W');
  do_xdata      = find(pl, 'xdata');
  do_ydata      = find(pl, 'ydata');
  do_zdata      = find(pl, 'zdata');

  %% Check the <char> option if it begins and ends  with a quote '
  %% If not then add the quote.
  if ischar(option) && ~isempty(option)
    if ~strcmp(option(1), '''')
      option = [''''' option];
    end
    if ~strcmp(option(end), '''')
      option = [option '''''];
    end
  end

  %% Convert the <double> option into a char
  if isnumeric(option) || islogical(option)
    option = num2str(option);
  end

  %% Convert the dim into a char
  if isnumeric(dim) || islogical(dim)
    dim = num2str(dim);
```

```
  end

  %% The operation can only compute with a 'option' or with a 'dimension'.
  %% e.g. operation(ao, 'option')
  %%      operation(ao, dim)
  if ~isempty(option) && ~isempty(dim)
    warning ('### xyzdata/single_operation: The param list contain a ''option'' and a ''dim''. It is computed %s(ao,option).', op);
  end

  %%
  if isempty(do_xdata) && isempty(do_ydata)
    do_ydata = 'y';
  end

%% %% Ensure that the called function have at least two output variables
  if (nargout >= 2)

%%  %% xdata:
    %% Create the output string for the eval command
    %% e.g.: x_new = '[ x_new1 x_new2 ]'
    x_new = '[';
    for i = 1:nargout-1
      new_var_name = genvarname (sprintf('x_new%d',i));
      eval ([new_var_name '= 0;']);

      x_new = strcat(x_new, sprintf (' x_new%d',i));
    end
    x_new = strcat (x_new, ' ]');

    %% ydata:
    %% Create the output string for the eval command
    %% e.g.: y_new = '[ y_new1 y_new2 ]'
    y_new = '[';
    for i = 1:nargout-1
      new_var_name = genvarname (sprintf('y_new%d',i));
      eval ([new_var_name '= 0;']);

      y_new = strcat(y_new, sprintf (' y_new%d',i));
    end
    y_new = strcat (y_new, ' ]');

%%  %% Create the output variables
    %% e.g.: data_out1 = data;
    for i = 1:nargout-1
      new_var_name = genvarname (sprintf('data_out%d',i));
      eval ([new_var_name '= data;']);
    end

%%  Execute the command for the xdata
    if ~isempty(do_xdata)

      try
```

```
        %% Execute with a option
        %% >> operation(x,'option');
        if ~isempty(option)
          cmd_line = sprintf('%s = %s(x,%s);',x_new, op, option);
          eval (cmd_line);

        %% Execute with a weight vector and dimension
        %% >> operation(x,W,dim);
        elseif ~isempty(dim) && ~isempty(weight_vector)
          cmd_line = sprintf('%s = %s(x,weight_vector,%s);',x_new, op, dim);
          eval (cmd_line);

        %% Execute with a dimension
        %% >> operation(x,dim);
        elseif ~isempty(dim)
          cmd_line = sprintf('%s = %s(x,%s);',x_new, op, dim);
          eval (cmd_line);

        %% Execute with a weight vector
        %% >> operation(x,W);
        elseif ~isempty(weight_vector)
          cmd_line = sprintf('%s = %s(x,weight_vector);',x_new, op);
          eval (cmd_line);

        %% Execute without parameter or option
        %% >> operation(x)
        else
          cmd_line = sprintf('%s = %s(x);',x_new, op);
          eval (cmd_line);
        end

      catch
        l_error = lasterror;
        disp(sprintf('\n\n%s', l_error.message));
        error('### cdata/single_operation: Can not command:\n###    %s\n### Have a look to the param list', cmd_line);
      end

    end

%%  Execute the command for the ydata
    if ~isempty(do_ydata)

      try
        %% Execute with a option or a dimension
        %% >> operation(y,'option');
        if ~isempty(option)
          cmd_line = sprintf('%s = %s(y,%s);',y_new, op, option);
          eval (cmd_line);

        %% Execute with a weight vector and dimension
        %% >> operation(y,W,dim);
        elseif ~isempty(dim) && ~isempty(weight_vector)
```

```
          cmd_line = sprintf('%s = %s(y,weight_vector,%s);',y_new, op, dim);
          eval (cmd_line);

        %% Execute with a option or a dimension
        %% >> operation(y,dim);
        elseif ~isempty(dim)
          cmd_line = sprintf('%s = %s(y,%s);',y_new, op, dim);
          eval (cmd_line);

        %% Execute with a weight vector
        %% >> operation(y,W);
        elseif ~isempty(weight_vector)
          cmd_line = sprintf('%s = %s(y,weight_vector);',y_new, op);
          eval (cmd_line);

        %% Execute without parameter or option
        %% >> operation(y)
        else
          cmd_line = sprintf('%s = %s(y);',y_new, op);
          eval (cmd_line);
        end

      catch
        l_error = lasterror;
        disp(sprintf('\n\n%s', l_error.message));
        error('### cdata/single_operation: Can not command:\n###    %s\n### Have a look to the param list', cmd_line);
      end

    end

%%  %% The result from eval is in x_new1, y_new1, ...
    %% Create a xydata object from this results and write it into the output
    for i = 1:nargout-1

      %% Create xydata
      if ~isempty(do_xdata)
        eval (sprintf('data_out%d.x = x_new%d;',i,i));
        eval (sprintf('data_out%d.name = ''%s(%s)'';',i,op,data.name));
        eval (sprintf('data_out%d.xunits = ''%s(%s)'';',i,op,data.xunits));
      end
      if ~isempty(do_ydata)
        eval (sprintf('data_out%d.y = y_new%d;',i,i));
        eval (sprintf('data_out%d.name = ''%s(%s)'';',i,op,data.name));
        eval (sprintf('data_out%d.yunits = ''%s(%s)'';',i,op,data.yunits));
      end

      %% Write to output
      eval (sprintf('varargout{i+1} = data_out%d;',i))

    end

    %% Create a new history object
```

```
    h = history(op, VERSION);

    varargout{1} = h;

  else % (nargout < 2)
    error ('### xydata/single_operation: This function need at least two output variables.');
  end

else
  error ('### xydata/single_operation: This function need three input variables.')
end

function pl_default = getDefaultPL()

  pl_default = plist('option', '',  ...
                     'dim',    '',  ...
                     'W',      [],  ...
                     'xdata',  '',  ...
                     'ydata',  'y');

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% path: m/noisegenerator %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%   path: m/noisegenerator   %%%%%%%%%%%%%%%%%%

   <a href="matlab:help m/noisegenerator/ltpda_noisegen">m/noisegenerator/
ltpda_noisegen</a> -  uses pzm2ab and franklin to generate a time-series
   <a href="matlab:help m/noisegenerator/ngconv">m/noisegenerator/ngconv</a>
- % get zeros and poles from pzmodels
   <a href="matlab:help m/noisegenerator/nginit">m/noisegenerator/nginit</a>
- % writing the generator
   m/noisegenerator/ngprop         - (No help available)
   <a href="matlab:help m/noisegenerator/ngsetup">m/noisegenerator/ngsetup</
a>        - ALGONAME = mfilename;
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 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

 VERSION:     $Id: hist2dot.m,v 1.4 2008/02/12 22:30:27 hewitson 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:

SOURCE CODE ^

```
function varargout = hist2dot(varargin)
% HIST2DOT converts a history object to a 'DOT' file suitable for processing with graphviz
%
%%%
```

```
% write header
fprintf(fd, 'digraph G \n{\n');

[n,a, nodes] = getNodes(h, [], 0, 1, []);

% Write block set
fprintf(fd, 'end [label="END"];\n');
for j=1:length(nodes)
  node = nodes(j);
  % wrap label
  % Do some command substitution
  fcn = node.names;
  shape = 'rectangle';
  fsize  = 12;
  Opsize = 24;
  opextras = 'style=filled, fillcolor=gray90 fixedsize=true width=0.5';
  extras  = '';
  switch fcn
    case 'times'
      fcn = '*';
      shape = 'circle';
      fsize = Opsize;
      extras = opextras;
    case 'mtimes'
      fcn = '.*';
      shape = 'circle';
      fsize = Opsize;
      extras = opextras;
    case 'plus'
      fcn = '+';
      shape = 'circle';
      fsize = Opsize;
      extras = opextras;
    case 'minus'
      fcn = '-';
      shape = 'circle';
      fsize = Opsize;
      extras = opextras;
    case 'rdivide'
      fcn = './';
      shape = 'circle';
      fsize = Opsize;
      extras = opextras;
    case 'mrdivide'
      fcn = '/';
      shape = 'circle';
      fsize = Opsize;
      extras = opextras;
  end

  wstr = 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);

```

Generated on Tue 26-Feb-2008 10:52:52 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  - ???

 PARAMETERS:  (Add these parameters to the default parameter list)
   A: - parameter description
   X: - parameter description
   DESCRIPTION: - user defined routine description

 VERSION:     $Id: method_fcn_template.m,v 1.3 2008/02/15 16:25:38 mauro Exp $

 HISTORY:     dd-mm-yyyy M Hewitson
                 Creation

 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')

%%%
```

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

```
%   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('foo(%s)', a.data.name));
  xy = set(xy, 'xunits', a.data.yunits);
  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('foo(%s)', invars{j}),...
    'mfilename', ALGONAME, ...
    'description', find(pl,'description'));

  bs = [bs b];

end


%% Reshape the ouput to the same size of the input
varargout{1} = reshape(bs, size(as));


%% %%%%
```

```
% 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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% path: m/noisegenerator/private %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   path: m/noisegenerator/private   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help m/noisegenerator/private/conv_noisegen">m/noisegenerator/
private/conv_noisegen</a> - normalize to bn = 1
   m/noisegenerator/private/fq2fac        - (No help available)
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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

 VERSION:     $Id: hist2m.m,v 1.17 2008/02/22 22:28:48 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:

SUBFUNCTIONS ^

- function cmd = writeCmd(name, pl, aon, ains)
- function ps = writePlist(pl)

SOURCE CODE ^

```
function cmds = hist2m(varargin)
% HIST2M writes a new m-file that reproduces the analysis described in the history object.
%
%%%
```

```
% the number of the last digits
CREATED2INT = 1e7;

% get a node list
disp(' -- extracting node list from history');
[n,a, nodes] = getNodes(h, [], 0, 1, []);

disp(' -- converting history nodes to commands');
% loop over nodes and convert to commands
i = 1;
cmds = {};
while i <= length(nodes)
  v      = nodes(i).n;
  idx    = find([nodes(:).pn] == i);
  pl     = nodes(i).pl;
  h      = nodes(i).h;
  aoName = mod(h.created.utc_epoch_milli,CREATED2INT);
  hi     = [nodes(idx).h];
  iNames = zeros(size(hi));
  for j=1:length(hi)
    created = hi(j).created; %get(hi(j), 'created');
    iNames(j) =  mod(created.utc_epoch_milli,CREATED2INT);
  end
  cmd = writeCmd(char(nodes(i).names), pl, aoName, iNames);
  disp(sprintf(' -- wrote command for node %d [%s]', i, char(nodes(i).names)));
  cmds = [cmds cellstr(cmd)];
  i = i + 1;
end

% now find commands that are duplicated after the '=' and remap those
disp(' -- fixing duplicate commands');
ncmds = length(cmds);
for j = 1:ncmds
  cmdj = cmds{j};
  % now inspect all other commands prior to this one
  for k = j+1:ncmds
    cmdk = cmds{k};
    if strcmp(cmdj, cmdk)
      cmds{j} = '';
    end
  end
end

disp(' -- writing output line');
% add the final command to produce a_out
alast = deblank(strtok(cmds{1}, '='));
cmds  = [cellstr(sprintf('a_out = %s;', alast)) cmds];

%-----------------------------------------------------------------------
%
% write a command-line
%
```

```
function cmd = writeCmd(name, pl, aon, ains)

ainsStr = '';
% for i=length(ains):-1:1
ni = length(ains);
for i=1:ni
  ainsStr = [ainsStr sprintf('a%d, ', ains(i))];
end

name = strrep(name, '\_', '_');

% look at the input parameters
if isa(pl, 'plist')
  ps = writePlist(pl);
else
  np = 0;
  ps = '';
end
if strcmp(ps, 'plist([])')
  ps = '';
end
pstr = deblank(sprintf('%s%s', ainsStr, ps));

if ~isempty(pstr)
  if pstr(end) == ','
    pstr = pstr(1:end-1);
  end
end
cmd = sprintf('a%d = %s(%s);', aon, name, pstr);


%-------------------------------------------------------
function ps = writePlist(pl)

ps = 'plist([';
np = length(pl.params(:));
for i=1:np
  p = pl.params(i);
  if ~isempty(p.val)

    %%%   char   %%%
    if ischar(p.val)
      ps = [ps sprintf('param(''%s'', ''%s'') ', p.key, p.val)];

    %%%   numeric   %%%
    elseif isnumeric(p.val)
      ps = [ps [sprintf('param(''%s'', [', p.key)  ltpda_mat2str(p.val) ']) ']];

    %%%   specwin   %%%
    elseif isa(p.val, 'specwin')
      w = p.val;
      wstr = string(w);
```

```
      ps = [ps sprintf('param(''%s'', %s) ', p.key, wstr) ];

    %%%   miir   %%%
    elseif isa(p.val, 'miir')
      f = p.val;
      fstr = string(f);
      ps = [ps sprintf('param(''%s'', %s) ', p.key, fstr) ];

    %%%   pzmodel   %%%
    elseif isa(p.val, 'pzmodel')
      pzm = p.val;
      fstr = string(pzm);
      ps = [ps sprintf('param(''%s'', %s) ', p.key, fstr) ];

    %%%   time   %%%
    elseif isa(p.val, 'time')
      tt = p.val;
      fstr = string(tt);
      ps = [ps sprintf('param(''%s'', %s) ', p.key, fstr) ];

    %%%   provenance   %%%
    elseif isa(p.val, 'provenance')

    %%%   history   %%%
    elseif isa(p.val, 'history')

    %%%   plist   %%%
    elseif isa(p.val, 'plist')
      fstr = string(p.val);
      ps = [ps sprintf('%s', fstr) ];
    else
      error(['### unknown parameter type: ' p.key '  ' class(p.val)]);
    end
  end
end
ps = [ps '])'];

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

mfiltload

PURPOSE ^

MFILTLOAD Load in a filter from *.filt files.

SYNOPSIS ^

function f = mfiltload(filename)

DESCRIPTION ^

```
MFILTLOAD Load in a filter from *.filt files.

usage:  f = mfiltload(filename)

M Hewitson 11-02-07


$Id: mfiltload.m,v 1.2 2007/10/24 10:55:11 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function f = mfiltload(filename)

% MFILTLOAD Load in a filter from *.filt files.
%
% usage:  f = mfiltload(filename)
%
% M Hewitson 11-02-07
%
%
% $Id: mfiltload.m,v 1.2 2007/10/24 10:55:11 ingo Exp $
%


delims = ' \n';
```

```
[in, msg] = fopen(filename, 'rt');

if(in < 0)
  error(['can not open file ' filename]);
end


n = 1;
ncoeffs = 1;
filt.a = [];
filt.b = [];
norminst = '';

lbuffer = fgets(in);

while (lbuffer > 0)
  if(length(lbuffer) > 1)
    % read the first token on this line
    [token, r] = strtok (lbuffer, delims);
    % ignore comments
    if(token(1) ~= '#')
      if(strcmp(token, 'NAME')) % name tag
        % get next token as name
        [token, r] = strtok (r, delims);
        if(length(token) > 0)
          filt.name = token;
        end
      elseif(strcmp(token, 'GAIN'))
        % get next token as gain
        [token, r] = strtok (r, delims);
        if(length(token) > 0)
          filt.gain = str2num(token);
          if(filt.gain < 0 | filt.gain > 1e20)
            error(['### gain is a strange number (' num2str(filt.gain) ')']);
          end
        end
      elseif(strcmp(token, 'FS'))
        % get next token as fs
        [token, r] = strtok (r, delims);
        if(length(token) > 0)
          filt.fs = str2num(token);
          if(filt.fs < 0 | filt.fs > 1e10)
            error(['### sample rate is a strange number (' num2str(filt.fs)
')']);
          end
        end
      elseif(strcmp(token, 'NTAPS'))
        % get next token as ntaps
        [token, r] = strtok (r, delims);
        if(length(token) > 0)
          filt.ntaps = str2num(token);
```

```
          if(filt.ntaps <= 0 | filt.ntaps > 20)
            error('### number of taps is out of bounds (1->10)');
          end
        end
      elseif(strcmp(token, 'NORM'))
        % get next token as normalisation instruction
        [token, r] = strtok (r, delims);
        norminst = token;
      elseif(strcmp(token, 'COEFF'))
        if(ncoeffs <= filt.ntaps)
          % get next token as A coeff
          [token, r] = strtok (r, delims);
          if(length(token) == 0)
            filt.a(ncoeffs) = 0;
          else
            filt.a(ncoeffs) = str2num(token);
          end

          % get next token as B coeff
          [token, r] = strtok (r, delims);
          if(length(token) == 0)
            filt.b(ncoeffs) = 0;
          else
            filt.b(ncoeffs) = str2num(token);
          end

          ncoeffs = ncoeffs + 1;
        else
          error(['### max taps reached. Ignoring line ' num2str(n) ]);
        end
      else
        warning(['### keyword not found (line ' num2str(n) ')' ]);
      end
    end
  end
  n = n + 1;
  lbuffer = fgets(in);
end

% close file
fclose(in);

% before returning, normalise gain if required
if(length(norminst) > 0)
    if(strncmp(norminst, 'dc', 2))
       % normalise dc to unity
       gain = iirdcgain(filt.a, filt.b);
       filt.a = filt.a ./ gain;
    elseif(strncmp(norminst, 'hf', 2))
       % normalise hf to unity
       gain = iirhfgain(filt.a, filt.b);
```

```
        filt.a = filt.a ./ gain;
    else
      error(['### readfilt: normalisation instruction (' norminst ') not recognised.']);
    end
end

f.name    = filt.name;
f.fs      = filt.fs;
f.ntaps   = filt.ntaps;
f.a       = filt.a;
f.b       = filt.b;
f.gain    = filt.gain;
f.histin  = zeros(1, f.ntaps-1);
f.histout = zeros(1, f.ntaps-1);


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% path: m/plottools %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   path: m/plottools   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help m/plottools/islinespec">m/plottools/islinespec</a>      -
checks a string to the line spec syntax.
   <a href="matlab:help m/plottools/ltpda_allgrid">m/plottools/ltpda_allgrid</a>
-  Set all the grids to 'state'
   <a href="matlab:help m/plottools/ltpda_allxaxis">m/plottools/ltpda_allxaxis</a>
- LTPDA_ALLXSCALE Set all the x scales on the current figure.
   <a href="matlab:help m/plottools/ltpda_allxlabel">m/plottools/ltpda_allxlabel</a>
-  Set all the x-axis labels on the current figure.
   <a href="matlab:help m/plottools/ltpda_allxscale">m/plottools/ltpda_allxscale</a>
-  Set all the x scales on the current figure.
   <a href="matlab:help m/plottools/ltpda_allyaxis">m/plottools/ltpda_allyaxis</a>
-  Set all the yaxis ranges on the current figure.
   <a href="matlab:help m/plottools/ltpda_allylabel">m/plottools/ltpda_allylabel</a>
-  Set all the y-axis labels on the current figure.
   <a href="matlab:help m/plottools/ltpda_allyscale">m/plottools/ltpda_allyscale</a>
-  Set all the Y scales on the current figure.
   <a href="matlab:help m/plottools/ltpda_cscale">m/plottools/ltpda_cscale</a>    -
Set the color range of the current figure
   <a href="matlab:help m/plottools/ltpda_legendAdd">m/plottools/ltpda_legendAdd</a>
-  Add a string to the current legend.
   <a href="matlab:help m/plottools/ltpda_msuptitle">m/plottools/ltpda_msuptitle</a>
-  Puts a title above all subplots.
   <a href="matlab:help m/plottools/ltpda_xaxis">m/plottools/ltpda_xaxis</a>     -
Set the X axis range of the current figure
   <a href="matlab:help m/plottools/ltpda_xscale">m/plottools/ltpda_xscale</a>    -
Set the X scale of the current axis
   <a href="matlab:help m/plottools/ltpda_xticks">m/plottools/ltpda_xticks</a>    -
set the input vector as the y-ticks of the current axis.
   <a href="matlab:help m/plottools/ltpda_yaxis">m/plottools/ltpda_yaxis</a>     -
Set the Y axis range of the current figure
   <a href="matlab:help m/plottools/ltpda_yscale">m/plottools/ltpda_yscale</a>    -
Set the Y scale of the current axis
   <a href="matlab:help m/plottools/ltpda_yticks">m/plottools/ltpda_yticks</a>    -
```

```
set the input vector as the y-ticks of the current axis.
   <a href="matlab:help m/plottools/ltpda_zaxis">m/plottools/ltpda_zaxis</a>     -
Set the Z axis range of the current figure
   <a href="matlab:help m/plottools/ltpda_zscale">m/plottools/ltpda_zscale</a>    -
Set the Z scale of the current axis
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

history

PURPOSE ^

HISTORY History object class constructor.

SYNOPSIS ^

function h = history(varargin)

DESCRIPTION ^

```
 HISTORY History object class constructor.

%%%
```

- history HISTORY History object class constructor.

This function is called by:

- history HISTORY History object class constructor.
- subsasgn SUBSASGN define index assignment for history properties.

SUBFUNCTIONS ^

- function h = init()

SOURCE CODE ^

```
function h = history(varargin)
% HISTORY History object class constructor.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: HISTORY History object class constructor.
%              Create a history object.
%
% CONSTRUCTOR: h = history()
%                  creates a blank history object
%
%              h = history('name', 'version')
%                  creates a history object with this name and version
%
%              h = history('name', 'version', plist)
%                  creates a history object with this name,
%                  version and parameter list.
%
%              h = history('name', 'version', plist, hi)
%                  creates a history object with this name, version,
%                  parameter list, and vector of input history objects.
%
%              h = history(hi)
%                  creates a new history object from the history object hi
%
% VERSION: $Id: history.m,v 1.19 2008/02/13 10:13:53 ingo Exp $
%
% HISTORY: 30-01-2007 M Hewitson
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: history.m,v 1.19 2008/02/13 10:13:53 ingo Exp $';
CATEGORY = 'Constructor';

%%%%%   Is this a 'Params' call?     %%%%%
if  nargin == 2 && isa(varargin{1}, 'history') && ischar(varargin{2})
```

```
  if strcmp(varargin{2}, 'Params')
    h = plist();
    return
  elseif strcmp(varargin{2}, 'Version')
    h = VERSION;
    return
  elseif strcmp(varargin{2}, 'Category')
    h = CATEGORY;
    return
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%   define history properties   %%%%%%%%%%%%%%%%%%%%%%%%

  function h = init()
    h.name    = 'None';
    h.version = VERSION;
    h.plist   = plist();
    h.inhists = [];
    h.invars  = {};
    h.n       = -1;
    h.pn      = -1;
    h.created = time;
    h.consver = VERSION;
    h = class(h, 'history');
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%   Create history object   %%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%  h = history()   %%%%%%%%%%
% create default history object
if nargin == 0

  h = init();

elseif nargin == 1

  %%%%%%%%%%%% From File %%%%%%%%%%%%%%%%
  if ischar(varargin{1})

    filename = varargin{1};
    [path, name, ext, vers] = fileparts(filename);
    switch ext
      case '.mat'
        h = load(filename);
        h = h.a;
      case '.xml'
        root_node = xmlread(filename);
        h = ltpda_xmlread(root_node, 'history');
      otherwise
        error('### Unknown file type.');
    end
  %%%%%%%%%%  h = history(struct)   %%%%%%%%%%
```

```
  elseif isstruct(varargin{1})

    h = init();

    hstruct = varargin{1};
    h.name    = hstruct.name;
    h.version = hstruct.version;
    pl = hstruct.plist;
    if isstruct(pl)
      pl = plist(pl);
    end
    h.plist   = pl;
    inh = hstruct.inhists;
    inhists = [];
    for j=1:length(inh)
      if isstruct(inh(j))
        inhists = [inhists history(inh(j))];
      else
        inhists = [inhists inh(j)];
      end
    end
    h.inhists = inhists;
    h.invars  = hstruct.invars;
    h.n       = hstruct.n;
    h.pn      = hstruct.pn;
    created   = hstruct.created;
    if isstruct(created)
      created = time(created);
    end
    h.created     = created;
    h.consver = hstruct.consver;

  %%%%%%%%%%  h = history(history)   %%%%%%%%%%
  else
    h = varargin{1};
    h.consver = VERSION;
  end

%%%%%%%%%%  h = history(name, version)   %%%%%%%%%%
elseif nargin == 2
  %%%%%%%%%%% Database
  if isa(varargin{1}, 'database')
    h = retrieve(varargin{1}, varargin{2:end});
  else
    h         = init();
    h.name    = varargin{1};
    h.version = varargin{2};
  end

%%%%%%%%%%  h = history(name, version, plist)   %%%%%%%%%%
elseif nargin == 3
  if isempty(varargin{3})
```

```
    varargin{3} = plist();
  end

  h         = init();
  h.name    = varargin{1};
  h.version = varargin{2};
  h.plist   = varargin{3};

%%%%%%%%%%  h = history(name, version, plist, history)   %%%%%%%%%%
elseif nargin == 4
  if isempty(varargin{3})
    varargin{3} = plist();
  end

  h         = init();
  h.name    = varargin{1};
  h.version = varargin{2};
  h.plist   = varargin{3};
  h.inhists = varargin{4};

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

end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

mfir

PURPOSE ^

MFIR FIR filter object class constructor.

SYNOPSIS ^

function f = mfir(varargin)

DESCRIPTION ^

```
MFIR FIR filter object class constructor.
    Create an mfir object.

  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(pl)   - create an mfir object from the description given in the
                   parameter list.

Parameters for standard filters:

             'type'   - 'highpass', 'lowpass',
                        'bandpass', 'bandreject'.
                        <<default: 'lowpass'>>
             'gain'   - gain of filter
                        <<default: 1.0>>
             'fs'     - sample frequency to design for
                        <<default: 1Hz>>
             'order'  - order of filter
                        <<default: 1+1>>
             'fc'     - corner frequencies.
                        This is a two element vector for bandpass
                        and bandreject filters.
                        <<default: 0.25 or [0.1 0.2]>>
             'win'    - Specify window function used in filter design
                        <<default: Hamming>>

Parameters for filter design from AO/fsdata:
  'method' - design method:
             'frequency-sampling' - uses fir2()
```

```
              'least-squares'      - uses firls()
              'Parks-McClellan'    - uses firpm()
   'Win'    - Window function for frequency-sampling method
   'N'      - filter order

 Parameters for retrieving from repository:

   'Hostname' - construct an AO by retrieving it from an LTPDA repository
                specified by the given hostname. Only those objects which
                are mfir objects are returned.
                Additional parameters:
                'Database'   - The database name [default: 'ltpda']
                'ID'         - A vector of object IDs.

 EXAMPLE 1:   Create a default lowpass filter.

              >> f = mfir(plist())

 EXAMPLE 2:   Create an order 1+1 highpass filter with high frequency gain 2.
              Filter is designed for 1kHz sampled data and has a cut-off
              frequency of 200Hz.

              >> pl = plist();
              >> pl = append(pl, param('type', 'highpass'))
              >> pl = append(pl, param('order', 1))
              >> pl = append(pl, param('gain', 2.0))
              >> pl = append(pl, param('fs', 1000))
              >> pl = append(pl, param('fc', 200))
              >> f = mfir(pl)


 VERSION:     $Id: mfir.m,v 1.26 2008/02/24 10:15:55 hewitson Exp $

 HISTORY:     09-02-2007 M Hewitson
                 Creation
              11-02-2008 M Hueller
                 Help fixed
                 Default parameter list redefined

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = mfir(mfir,'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = mfir(mfir,'Version')

 The following call returns a string that contains the routine category:

 >> category = mfir(mfir,'Category')

%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

- get GET get mfir properties.
- mfir MFIR FIR filter object class constructor.
- set SET set a mfir object property.

This function is called by:

- mfir MFIR FIR filter object class constructor.
- redesign REDESIGN redesign the input filter to work for the given sample rate.
- subsasgn SUBSASGN define index assignment for mfir properties.

SUBFUNCTIONS ^

- function fs = mfirFromRepository(pl, VERSION, ALGONAME)
- function filt = mfirFromAO(ALGONAME, VERSION, a, pl)
- function pl = getDefaultPlist(varargin)
- function f = init(VERSION)

SOURCE CODE ^

```
function f = mfir(varargin)
% MFIR FIR filter object class constructor.
%     Create an mfir object.
%
%   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(pl)   - create an mfir object from the description given in the
%                    parameter list.
%
% Parameters for standard filters:
%
%              'type'   - 'highpass', 'lowpass',
%                         'bandpass', 'bandreject'.
%                         <<default: 'lowpass'>>
%              'gain'   - gain of filter
%                         <<default: 1.0>>
%              'fs'     - sample frequency to design for
%                         <<default: 1Hz>>
%              'order'  - order of filter
```

```
%                          <<default: 1+1>>
%              'fc'     - corner frequencies.
%                         This is a two element vector for bandpass
%                         and bandreject filters.
%                         <<default: 0.25 or [0.1 0.2]>>
%              'win'    - Specify window function used in filter design
%                         <<default: Hamming>>
%
% Parameters for filter design from AO/fsdata:
%   'method' - design method:
%              'frequency-sampling' - uses fir2()
%              'least-squares'      - uses firls()
%              'Parks-McClellan'    - uses firpm()
%   'Win'    - Window function for frequency-sampling method
%   'N'      - filter order
%
% Parameters for retrieving from repository:
%
%   'Hostname' - construct an AO by retrieving it from an LTPDA repository
%                specified by the given hostname. Only those objects which
%                are mfir objects are returned.
%                Additional parameters:
%                'Database'   - The database name [default: 'ltpda']
%                'ID'         - A vector of object IDs.
%
% EXAMPLE 1:   Create a default lowpass filter.
%
%              >> f = mfir(plist())
%
% EXAMPLE 2:   Create an order 1+1 highpass filter with high frequency gain 2.
%              Filter is designed for 1kHz sampled data and has a cut-off
%              frequency of 200Hz.
%
%              >> pl = plist();
%              >> pl = append(pl, param('type', 'highpass'))
%              >> pl = append(pl, param('order', 1))
%              >> pl = append(pl, param('gain', 2.0))
%              >> pl = append(pl, param('fs', 1000))
%              >> pl = append(pl, param('fc', 200))
%              >> f = mfir(pl)
%
%
% VERSION:     $Id: mfir.m,v 1.26 2008/02/24 10:15:55 hewitson Exp $
%
% HISTORY:     09-02-2007 M Hewitson
%                 Creation
%              11-02-2008 M Hueller
%                 Help fixed
%                 Default parameter list redefined
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
```

```
% >> pl = mfir(mfir,'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = mfir(mfir,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = mfir(mfir,'Category')
%
%%%
```

```
    if ischar(varargin{1})

      filename = varargin{1};
      [path, name, ext, vers] = fileparts(filename);
      switch ext
        case '.mat'
          f = load(filename);
          f = f.a;
        case '.xml'
          root_node = xmlread(filename);
          f = ltpda_xmlread(root_node, 'mfir');
        otherwise
          error('### Unknown file type.');
      end
      %%%%%%%%%%  f = mfir(mfir)   %%%%%%%%%%
    elseif isa(varargin{1}, 'mfir')
      f = varargin{1};
      f.version = VERSION;

      %%%%%%%%%%  f = mfir(ao)   %%%%%%%%%%
    elseif isa(varargin{1}, 'ao')
      pl = getDefaultPlist('From AO');
      f  = mfirFromAO(ALGONAME, VERSION, varargin{1}, pl);

      %%%%%%%%%%  f = mfir(struct)   %%%%%%%%%%
    elseif isstruct(varargin{1})

      f = init(VERSION);

      fields = fieldnames(varargin{1});
      for ii = 1:length(fields)
        field = fields{ii};

        %%% plist -> plist-object
        if strcmp(field, 'plist')
          if isstruct(varargin{1}.(field))
            f.(field) = plist(varargin{1}.(field));
          else
            f.(field) = varargin{1}.(field);
          end
          %%% created -> time-object
        elseif strcmp(field, 'created')
          created = varargin{1}.created;
          if isstruct(created)
            created = time(created);
          end
          f.created     = created;
          %%% All other
        else
          try
            f.(field) = varargin{1}.(field);
          catch
```

```
            error('### The field ''%s'' in the struct is not a mfir property.', field)
          end
        end
      end

      %%%%%%%%%%  f = mfir(plist)   %%%%%%%%%%
      % Parameter list
    elseif isa(varargin{1}, 'plist')

      pl       = varargin{1};
      type     = find(pl, 'type');
      pzm      = find(pl, 'pzmodel');
      hostname = find(pl, 'hostname');

      f = init(VERSION);

      % deal with empty plist
      if isempty(type) && isempty(pzm) && isempty(hostname)
        type = 'lowpass';
      end

      if ~isempty(type)
        % check and fill parameter list
        plo = parseFilterParams(pl);
        switch type
          case 'lowpass'
            f = mklowpass(f, plo);
          case 'highpass'
            f = mkhighpass(f, plo);
          case 'bandpass'
            f = mkbandpass(f, plo);
          case 'bandreject'
            f = mkbandreject(f, plo);
          otherwise
            error('### unknown standard filter type in mfir constructor.');
        end
        f.infile  = '';
        f.plist   = plo;
        f.created = time;
        f.version = VERSION;
      elseif ~isempty(pzm)

        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=%2.2fHz.', fs)...
            sprintf('\nThe filter will be redesigned later when used.')]);

        end
        % make MFIR filter
```

```
        f = tomfir(pzm, plist('fs', fs));
        f = set(f, 'name', get(pzm, 'name'));
        f = set(f, 'plist', pl);

      elseif ~isempty(hostname)

        f = mfirFromRepository(pl, VERSION, ALGONAME);
        % remove connection if it's there
        f.plist = remove(pl, 'conn');

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

  case 2 % pzmodel followed by plist

    % From spectrum
    %%%%%%%%%%% From DATABASE
    if isa(varargin{1}, 'database')
      f = retrieve(varargin{1}, varargin{2:end});

      %%%%%%%%%%  f = mfir(pzmodel, plist)   %%%%%%%%%%
    elseif isa(varargin{1}, 'pzmodel')
      pzm = varargin{1};
      if ~isa(pzm, 'pzmodel')
        error('### unknown constructor type for mfir.');
      end
      pl = varargin{2};
      if ~isa(pl, 'plist')
        error('### unknown constructor type for mfir.');
      end

      pl = append(pl, param('pzmodel', pzm));

      f = mfir(pl);

      %%%%%%%%%%  f = mfir(ao, plist)   %%%%%%%%%%
    elseif isa(varargin{1}, 'ao')

      if isa(varargin{2}, 'plist')
        pl = combine(varargin{2}, getDefaultPlist('From AO'));
      else
        pl = getDefaultPlist('From AO');
      end
      f = mfirFromAO(ALGONAME, VERSION, varargin{1}, pl);

      %%%%%%%%%%%% From a coeffs %%%%%%%%%%%%%
    elseif isnumeric(varargin{1})
      a      = varargin{1};
```

```
      fs     = varargin{2};

      % Checking the coefficients are listed in rows
      if size(a,1)~=1
        a = a';
      end

      f         = init(VERSION);
      f.name    = 'from coeffs';
      f.fs      = fs;
      f.ntaps   = length(a);
      f.a       = a;
      f.gd      = (f.ntaps)/2;
      f.gain    = 1;
      f.histout = zeros(1,f.ntaps-1);   % initialise output history

    else
      error('### Unknown constructor method.');
    end
  otherwise
    error('### incorrect input arguments for mfir constructor.');
end

end


%-----------------------------------------------------------------------
% construct an mfir from a repository
%
function fs = mfirFromRepository(pl, VERSION, ALGONAME)

dpl = getDefaultPlist('From Repository');
pl  = combine(pl, dpl);

% 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);
```

```
fs  = [];
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 mfir
  if strcmp(tt, mfilename)
    %---- call database constructor
    a = ltpda_obj_retrieve(conn, id);
    %---- Add history
    %---- Add to output array
    fs = [fs a];
  else
    warning('    !skipping ID %d, type %s', id, tt);
  end

end

% close connection
if closeConn
  close(conn);
end

end

%-----------------------------------------------------------------------------
% create FIR filter from magnitude of input AO/fsdata
function filt = mfirFromAO(ALGONAME, VERSION, a, pl)

%% 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

N      = find(pl, 'N');
win    = find(pl, 'Win');
method = find(pl, 'method');

fs = a.data.fs;
f  = a.data.x;
xx = abs(a.data.y);
g  = 1; %median(xx);

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.name, 'Kaiser') || strcmp(win.name, 'Flattop')
        win = specwin(win.name, N+1, win.psll);
      else
        win = specwin(win.name, N+1);
      end
    end
    disp('** 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
    disp('** 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.');
    disp('** designing filter using Parks-McClellan method [help firpm]');
    mtaps = firpm(N, ffm, xx);
  otherwise
    error('### unknown filter design method.');
end

% Make mfir object
filt          = init(VERSION);
filt.name     = sprintf('fir(%s)', a.name);
filt.fs       = fs;
filt.ntaps    = length(mtaps);
filt.a        = mtaps;
filt.gd       = (filt.ntaps+1)/2;
filt.gain     = g;
filt.histout  = zeros(1,filt.ntaps-1);   % initialise output history
filt.infile   = '';
filt.plist    = pl;
filt.created  = time;
filt.version  = VERSION;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPL
%
% DESCRIPTION: Get default params
%
% HISTORY:     11-02-2008 M Hueller
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function pl = getDefaultPlist(varargin)
```

```
% list of available parameter sets
sets = {'Standard Type', 'From AO', 'From pzmodel', 'From Repository',...
  'From Plist'};

if nargin == 0
  pl = sets;
  return
end

set = varargin{1};

switch set
  case 'From Repository'
    pl = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
  case 'Standard Type'
    pl = plist('type','lowpass',...
      'gain',  1.0, ...
      'fs', 1.0, ...
      'order', 1, ...
      'fc', 0.1, ...
      'win', 'Hamming');
  case 'From AO'
    N  = 512;
    pl = plist(param('N', N));
    pl = append(pl, param('method', 'frequency-sampling'));
    pl = append(pl, param('Win', specwin('Hanning', N+1)));
  case 'From pzmodel'
    pl = plist('pzmodel', []);
  case 'From Plist'
    pl = plist('Plist', []);
  otherwise
    pl = plist();
end

end

%%%%%%%%%%%%%%%%%%%%%%%%   define mfir properties   %%%%%%%%%%%%%%%%%%%%%%%%

function f = init(VERSION)
f.name    = 'None';
f.fs      = 0;
f.ntaps   = 0;
f.a       = [];
f.gd      = 0;
f.gain    = 0;
f.histout = [];
f.infile  = '';
f.plist   = plist();
f.created = time;
f.version = VERSION;
f         = class(f, 'mfir');
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%   Create mfir object   %%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Generated on Tue 26-Feb-2008 10:52:52 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_compute_f">m/sigproc/
frequency_domain/ltpda_compute_f</a>  -  compute the frequency and resolution
vectors for use with mlpsd.
   <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 weighted transfer-function
estimation computed on a log frequency axis.
   <a href="matlab:help m/sigproc/frequency_domain/ltpda_lxspec">m/sigproc/
frequency_domain/ltpda_lxspec</a>     -  performs cross-spectral analysis of various
forms.
   <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_psd">m/sigproc/
frequency_domain/ltpda_psd</a>        -  a psd estimation based on pwelch from
```

```
MATLAB.
   <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/ltpda_xspec">m/sigproc/
frequency_domain/ltpda_xspec</a>      -  performs cross-spectral analysis of various
forms.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

miir

PURPOSE ^

MIIR IIR filter object class constructor.

SYNOPSIS ^

function f = miir(varargin)

DESCRIPTION ^

```
 MIIR IIR filter object class constructor.

%%%
```

```
                        <<default: 1>>
            'fc'     - corner frequencies.
                       This is a two element vector for bandpass
                       and bandreject filters.
                       <<default: 0.25 or [0.1 0.2]>>
            'ripple' - Specify pass/stop-band ripple for bandpass
                       and bandreject filters
                       <<default: 0.5>>

EXAMPLE 1:   Create a default lowpass filter.

             >> f = miir(plist())

EXAMPLE 2:   Create an order 1 highpass filter with high frequency gain 2.
             Filter is designed for 1kHz sampled data and has a cut-off
             frequency of 200Hz.

             >> pl = plist();
             >> pl = append(pl, param('type', 'highpass'))
             >> pl = append(pl, param('order', 1))
             >> pl = append(pl, param('gain', 2.0))
             >> pl = append(pl, param('fs', 1000))
             >> pl = append(pl, param('fc', 200))
             >> f = miir(pl)

NOTES:
        ** The convention used here for naming the filter coefficients is
           the opposite to MATLAB's convention. The rescursion 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)


VERSION:     $Id: miir.m,v 1.30 2008/02/24 10:15:55 hewitson Exp $

HISTORY:     09-02-2007 M Hewitson
                Creation
             11-02-2008 M Hueller
                Help fixed
                Default parameter list defined

The following call returns a parameter list object that contains the
default parameter values:

>> pl = miir(miir,'Params')

The following call returns a string that contains the routine CVS version:

>> version = miir(miir,'Version')

The following call returns a string that contains the routine category:
```

```
 >> category = miir(miir,'Category')

%%%
```

```
%              f = miir(pl)
%                  create an miir object from the description
%                  given in the parameter list.
%
% Parameters for standard filters:
%
%              'type'   - 'highpass', 'lowpass',
%                         'bandpass', 'bandreject'.
%                         <<default: 'lowpass'>>
%              'gain'   - gain of filter
%                         <<default: 1.0>>
%              'fs'     - sample frequency to design for
%                         <<default: 1Hz>>
%              'order'  - order of filter
%                         <<default: 1>>
%              'fc'     - corner frequencies.
%                         This is a two element vector for bandpass
%                         and bandreject filters.
%                         <<default: 0.25 or [0.1 0.2]>>
%              'ripple' - Specify pass/stop-band ripple for bandpass
%                         and bandreject filters
%                         <<default: 0.5>>
%
% EXAMPLE 1:   Create a default lowpass filter.
%
%              >> f = miir(plist())
%
% EXAMPLE 2:   Create an order 1 highpass filter with high frequency gain 2.
%              Filter is designed for 1kHz sampled data and has a cut-off
%              frequency of 200Hz.
%
%              >> pl = plist();
%              >> pl = append(pl, param('type', 'highpass'))
%              >> pl = append(pl, param('order', 1))
%              >> pl = append(pl, param('gain', 2.0))
%              >> pl = append(pl, param('fs', 1000))
%              >> pl = append(pl, param('fc', 200))
%              >> f = miir(pl)
%
% NOTES:
%         ** The convention used here for naming the filter coefficients is
%            the opposite to MATLAB's convention. The rescursion 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)
%
%
% VERSION:     $Id: miir.m,v 1.30 2008/02/24 10:15:55 hewitson Exp $
%
% HISTORY:     09-02-2007 M Hewitson
%                 Creation
%              11-02-2008 M Hueller
```

```
%                 Help fixed
%                 Default parameter list defined
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = miir(miir,'Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = miir(miir,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = miir(miir,'Category')
%
%%%
```

```
    f.infile  = '';
    f.plist   = plist();
    f.created = time;
    f.version = VERSION;
    f = class(f, 'miir');
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%   Create miir object   %%%%%%%%%%%%%%%%%%%%%%%%%%

switch nargin

  %%%%%%%%%%  h = miir()   %%%%%%%%%%
  % create default miir object
  case 0
    f = init();

  % Copy, LISO, filt, standard
  case 1

    %%%%%%%%%%%%% From File %%%%%%%%%%%%%
    if ischar(varargin{1})

      f = init();
      filename = varargin{1};
      [path, name, ext, vers] = fileparts(filename);
      switch ext
        case '.mat'
          f = load(filename);
          f = f.a;
        case '.xml'
          root_node = xmlread(filename);
          f = ltpda_xmlread(root_node, 'miir');
        case '.filt'
          error('### .filt files are no longer supported.');
        case '.fil'
          filt       = filload(filename);
          f.name     = filt.name;
          f.fs       = filt.fs;
          f.ntaps    = filt.ntaps;
          f.a        = filt.a;
          f.b        = filt.b;
          f.gain     = filt.gain;
          f.histin   = filt.histin;
          f.histout  = filt.histout;
          f.infile   = filename;
        otherwise
          error('### Unknown file type.');
      end

    %%%%%%%%%%  h = miir(miir)   %%%%%%%%%%
    % copy
    elseif isa(varargin{1}, 'miir')
      f = varargin{1};
```

```
      f.version = VERSION;

    %%%%%%%%%%  h = miir(struct)   %%%%%%%%%%
    % Struct
    elseif isstruct(varargin{1})

      f = init();

      fields = fieldnames(varargin{1});
      for ii = 1:length(fields)
        field = fields{ii};

        %%% plist -> plist-object
        if strcmp(field, 'plist')
          if isstruct(varargin{1}.(field))
            f.(field) = plist(varargin{1}.(field));
          else
            f.(field) = varargin{1}.(field);
          end
        %%% created -> time-object
        elseif strcmp(field, 'created')
          created = varargin{1}.created;
          if isstruct(created)
            created = time(created);
          end
          f.created     = created;
        %%% All other
        else
          try
            f.(field) = varargin{1}.(field);
          catch
            error('### The field ''%s'' in the struct is not a mfir property.', field)
          end
        end
      end


    %%%%%%%%%%  h = miir(plist)   %%%%%%%%%%
    % Parameter list
    elseif isa(varargin{1}, 'plist')

      pl   = varargin{1};
      type = find(pl, 'type');
      pzm  = find(pl, 'pzmodel');
      hostname = find(pl, 'hostname');

      f = init();

      % deal with empty plist
      if isempty(type) && isempty(pzm) && isempty(hostname)
        type = 'lowpass';
      end
```

```

      if ~isempty(type)
        % check and fill parameter list
        plo = parseFilterParams(pl);
        switch type
          case 'lowpass'
            f = mklowpass(f, plo);
          case 'highpass'
            f = mkhighpass(f, plo);
          case 'bandpass'
            f = mkbandpass(f, plo);
          case 'bandreject'
            f = mkbandreject(f, plo);
          otherwise
            error('### unknown standard filter type in miir constructor.');
        end
        f.infile  = '';
        f.plist   = plo;

      elseif ~isempty(pzm)

        f = init();

        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=%2.2fHz.', fs)...
                   sprintf('\nThe filter will be redesigned later when used.')]);

        end
        % make MIIR filter
        f = tomiir(pzm, fs);
        f = set(f, 'name', get(pzm, 'name'));
        f = set(f, 'plist', pl);

      elseif ~isempty(hostname)

        f = miirFromRepository(pl, VERSION, ALGONAME);
        f.plist = remove(pl, 'conn');

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

  %%%%%%%%%%  h = miir(pzmodel, plist)   %%%%%%%%%%
  case 2 % pzmodel followed by plist
```

```

  %%%%%%%%%% From DATABASE
  if isa(varargin{1}, 'database')
    f = retrieve(varargin{1}, varargin{2:end});
  else
    pzm = varargin{1};
    if ~isa(pzm, 'pzmodel')
      error('### unknown constructor type for miir.');
    end
    pl = varargin{2};
    if ~isa(pl, 'plist')
      error('### unknown constructor type for miir.');
    end

    pl = append(pl, param('pzmodel', pzm));

    f = miir(pl);
  end

  %%%%%%%%%%  h = miir(a, b, fs)   %%%%%%%%%%
  case 3

    % then we have 3 numeric inputs for a and b coefficients and fs
    a  = varargin{1};
    b  = varargin{2};
    fs = varargin{3};

    if ~isnumeric(a) || ~isnumeric(b) || ~isnumeric(fs)
      error('### unknown constructor type for miir.');
    end

    % Checking the coefficients are listed in rows
    if size(a,1)~=1
        a = a';
    end
    if size(b,1)~=1
        b = b';
    end

    f         = init();
    f.name    = 'AB';
    f.fs      = fs;
    f.ntaps   = length(a);
    f.a       = a;
    f.b       = b;
    f.gain    = 1;
    f.histin  = zeros(1,f.ntaps-1);       % initialise input history
    f.histout = zeros(1,f.ntaps-1);       % initialise output history

  otherwise
    error('### incorrect input arguments for miir constructor.');
end

```

```
end % function f = miir(varargin)


%-----------------------------------------------------------------------
% construct an miir from a repository
%
function fs = miirFromRepository(pl, VERSION, ALGONAME)

dpl = getDefaultPlist('From Repository');
pl  = combine(pl, dpl);

% 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);
fs  = [];
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 miir
  if strcmp(tt, mfilename)
    %---- call database constructor
    a = ltpda_obj_retrieve(conn, id);
    %---- Add history
    %---- Add to output array
    fs = [fs a];
  else
    warning('    !skipping ID %d, type %s', id, tt);
  end

end

% close connection
if closeConn
```

```
  close(conn);
end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPL
%
% DESCRIPTION: Get default params
%
% HISTORY:     11-02-2008 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function pl = getDefaultPlist(varargin)


% list of available parameter sets
sets = {'Standard Type', 'From pzmodel', 'From Repository',...
  'From Plist'};

if nargin == 0
  pl = sets;
  return
end

set = varargin{1};

switch set
  case 'From Repository'
    pl = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
  case 'Standard Type'
    pl = plist('type','lowpass',...
      'gain',  1.0, ...
      'fs', 1.0, ...
      'order', 1, ...
      'fc', 0.1, ...
      'ripple', 0.5);
  case 'From pzmodel'
    pl = plist('pzmodel', []);
  case 'From Plist'
    pl = plist('Plist', []);
  otherwise
    pl = plist();
end

end

```

Generated on Tue 26-Feb-2008 10:52:52 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 1.7 2008/02/26 08:54:19 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

- muitable UITABLE creates a two dimensional graphic uitable component in a figure window.

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 1.7 2008/02/26 08:54:19 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});


if size(fields,2) ~= size(results,2)
  fields = cell(1,size(results,2));
end

```

```
%    [mtable, buttons] = createTable (pnContainer, headers, data, buttonsFlag, 'PropName',PropValue, ...)
% [t,b] = createTable(gui.main, fields, results, false, ...
%   'AutoResizeMode',javax.swing.JTable.AUTO_RESIZE_OFF,...
%                    'Editable',false,'Position',[.01,.08,1-0.02,.73])

% add table
Tleft   = leftMargin;
Twidth  = Width - 2*leftMargin;
Theight = 500;
Tbottom = leftMargin + btnHeight + 20;

% sometimes the user wrote the SQL query by hand
% in which case getting the selected fields as the
% column headings fails. Instead we fix this for now
% by making empty (A-Z) column headings.

t = muitable(gui.main, results, fields, 'Position',...
  [Tleft Tbottom Twidth Theight]);
set(t, 'Editable', 0);
% get(t.Table)
t.Table.setAutoResizeMode(t.Table.AUTO_RESIZE_OFF);
% get(t.Table)
% cm = get(t.Table.getColumnModel)
% get(cm.Columns(1))

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]);

%-----------------------------------------------------------------------
% 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   = findobj('Tag', 'QueryString');
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 Tue 26-Feb-2008 10:52:52 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_chkts">m/sigproc/time_domain/
ltpda_chkts</a>         -  checks that the time-series in the input AO(s) are evenly
sampled according to the sample rate and an allowed error.
   <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_timedomainfit">m/sigproc/
time_domain/ltpda_timedomainfit</a> -  uses lscov to fit a set of time-series AOs to
a target time-series AO.
   <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
   <a href="matlab:help m/sigproc/time_domain/polydetrend">m/sigproc/time_domain/
polydetrend</a>         -  detrends the input data vector with a polynomial.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% path: m/timetools %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%   path: m/timetools   %%%%%%%%%%%%%%%%%%

   <a href="matlab:help m/timetools/ltpda_gps2utc">m/timetools/ltpda_gps2utc</
a>       -  converts GPS seconds to UTC time.
   <a href="matlab:help m/timetools/ltpda_reformat_date">m/timetools/
ltpda_reformat_date</a> -  reformats the input date
   <a href="matlab:help m/timetools/ltpda_utc2gps">m/timetools/ltpda_utc2gps</
a>       -  Converts UTC time to GPS seconds.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

minus

PURPOSE ^

MINUS overloads - operator for time objects.

SYNOPSIS ^

function res = minus(t1, t2)

DESCRIPTION ^

```
 MINUS overloads - operator for time objects.

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

 DESCRIPTION: MINUS overloads - operator for analysis objects.
              This function subtracts a time in seconds from a time object.
              It is possible to define the subtrahend as a string,
              number or a time object.

 CALL:        t2  = t1 - value (in seconds)
              sec = t1 - t2;

 EXAMPLES:    >> t1      = time('2007-07-01 12:23:33');
              >> t2      = t1 - 30;
              >> t3      = t1 - '30';
              >> seconds = t1 - t2;

 VERSION:     $Id: minus.m,v 1.6 2008/02/15 16:26:08 mauro Exp $

 HISTORY:     03-08-2007 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a time object into a string.
- set SET set time properties.

This function is called by:

SOURCE CODE ^

```
function res = minus(t1, t2)
% MINUS overloads - operator for time objects.
%
%%%
```

```
%%%%%   >> t2 = t1 - 30;   %%%%%
%%%%%   >> t2 = t1 - '30';   %%%%%
if ischar(t2)
  java_format_str_seconds = 'ss';

  utc_timezone   = java.util.TimeZone.getTimeZone('UTC');

  t_format_seconds = java.text.SimpleDateFormat(java_format_str_seconds);
  t_format_seconds.setTimeZone(utc_timezone);

  epoch_time_seconds = t_format_seconds.parse(t2);
  epoch_time_seconds = epoch_time_seconds.getTime;

  res = set(t1, 'utc_epoch_milli', t1.utc_epoch_milli-epoch_time_seconds);

%%%%%   >> number = t1 - t2;   %%%%%
elseif isa(t2, 'time')

  res = (t1.utc_epoch_milli - t2.utc_epoch_milli)/1000;

else
  error ('### Unknown minus-operation to a time object.');
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

abcascade

PURPOSE ^

Cascade two filters together to get a new filter.

SYNOPSIS ^

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

DESCRIPTION ^

```
Cascade two filters together to get a new filter.

usage: [a,b] = abcascade(a1,b1,a2,b2)

M Hewitson 03-04-07

$Id: abcascade.m,v 1.1 2007/04/03 11:34:23 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- pzm2ab PZM2AB convert pzmodel to IIR filter coefficients using bilinear

SOURCE CODE ^

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

% Cascade two filters together to get a new filter.
%
```

```
% usage: [a,b] = abcascade(a1,b1,a2,b2)
%
% M Hewitson 03-04-07
%
% $Id: abcascade.m,v 1.1 2007/04/03 11:34:23 hewitson Exp $
%

n = length(a1);
m = length(a2);
N = m+n-1;

alpha = zeros(1, N);
beta = zeros(1, N);

for i=0:n-1
  for j=0:m-1
    alpha(i+j+1) = alpha(i+j+1) + a1(i+1)*a2(j+1);
    beta(i+j+1) = beta(i+j+1) + b1(i+1)*b2(j+1);
  end
end

ao = alpha;
bo = beta;
```

Generated on Tue 26-Feb-2008 10:52:52 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.

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

 DESCRIPTION: MKBANDPASS return a bandpass filter miir().
              A Cheby filter is used.

 CALL:        f = mkbandpass(f, pl)

 VERSION: $Id: mkbandpass.m,v 1.4 2007/10/24 10:55:11 ingo Exp $

 HISTORY: 27-08-2002 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function f = mkbandpass(f, pl)
% MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: MKBANDPASS return a bandpass filter miir().
%              A Cheby filter is used.
%
```

```
% CALL:        f = mkbandpass(f, pl)
%
% VERSION: $Id: mkbandpass.m,v 1.4 2007/10/24 10:55:11 ingo Exp $
%
% HISTORY: 27-08-2002 M Hewitson
%             Creation
%
%%%
```

mkbandpass

PURPOSE ^

MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.

SYNOPSIS ^

function f = mkbandpass(f, pl)

DESCRIPTION ^

```
 MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.

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

 DESCRIPTION: MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.

 CALL:        f = mkbandpass(f, pl)

 VERSION: $Id: mkbandpass.m,v 1.5 2007/10/24 10:55:11 ingo Exp $

 HISTORY: 27-08-2002 M Hewitson
            Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function f = mkbandpass(f, pl)
% MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.
%
% CALL:        f = mkbandpass(f, pl)
```

```
%
% VERSION: $Id: mkbandpass.m,v 1.5 2007/10/24 10:55:11 ingo Exp $
%
% HISTORY: 27-08-2002 M Hewitson
%             Creation
%
%%%
```

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 2007/10/24 10:55:11 ingo Exp $

 HISTORY: 27-08-2002 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function f = mkbandreject(f, pl)
% 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 2007/10/24 10:55:11 ingo Exp $
%
% HISTORY: 27-08-2002 M Hewitson
%             Creation
%
%%%
```

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input history object.

SYNOPSIS ^

function cmd = string(hs, 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

 VERSION:     $Id: string.m,v 1.4 2008/02/12 22:30:27 hewitson Exp $

 HISTORY:     29-03-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- get GET get history object properties.
- 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.
- string STRING writes a command string that can be used to recreate the input history object.

SOURCE CODE ^

```
function cmd = string(hs, varargin)
% STRING writes a command string that can be used to recreate the input history object.
%
%%%
```

```
  hi   = get(h, 'inhists');

  if isempty(pl)
    plstr = '[]';
  else
    plstr = string(pl);
  end
  if isempty(plstr)
    plstr = '[]';
  end

  if isempty(hi)
    histr = '[]';
  else
    histr = string(hi);
  end
  if isempty(histr)
    histr = '[]';
  end

  cmd = [cmd 'history(' ...
             '''' name '''' ', ' ...
             '''' vers '''' ', ' ...
            plstr ', '  ...
            histr  ...
            ') '];
end

cmd = [cmd ']'];

if strcmp(cmd, '[]')
  cmd = '';
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(cdata, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(cdata, 'Version')

              The following call returns the routine category:
              >> cat = isfield(cdata, 'Category')

 VERSION:     $Id: isfield.m,v 1.4 2008/02/15 16:27:15 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)
% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 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 1.7 2008/01/02 17:54:22 ingo Exp $

 HISTORY: 27-08-2002 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function f = mkbandreject(f, pl)
% 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 1.7 2008/01/02 17:54:22 ingo Exp $
%
% HISTORY: 27-08-2002 M Hewitson
%             Creation
%
%%%
```

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input filter object.

SYNOPSIS ^

function cmd = string(f, 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)

 VERSION:     $Id: string.m,v 1.4 2008/02/12 22:39:49 hewitson Exp $

 HISTORY:     29-03-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a mfir object into a string.
- get GET get mfir properties.
- string STRING writes a command string that can be used to recreate the input filter object.

This function is called by:

- string STRING writes a command string that can be used to recreate the input filter object.

SOURCE CODE ^

```
function cmd = string(f, varargin)
% STRING writes a command string that can be used to recreate the input filter
```

```
object.
%
%%%
```

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(fsdata, 'Params')

              The following call returns the routine CVS version:
              >> version = isfield(fsdata, 'Version')

              The following call returns the routine category:
              >> category = isfield(fsdata, 'Category')

 VERSION:     $Id: isfield.m,v 1.4 2008/02/13 13:47:53 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)
% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 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 2007/10/24 10:55:11 ingo Exp $

 HISTORY: 27-08-2002 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function f = mkhighpass(f, pl)
% 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 2007/10/24 10:55:11 ingo Exp $
%
% HISTORY: 27-08-2002 M Hewitson
%             Creation
%
%%%
```

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input filter object.

SYNOPSIS ^

function cmd = string(f, 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

 VERSION:     $Id: string.m,v 1.4 2008/02/12 22:46:55 hewitson Exp $

 HISTORY:     29-03-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a miir object into a string.
- get GET get miir properties.
- string STRING writes a command string that can be used to recreate the input filter object.

This function is called by:

- string STRING writes a command string that can be used to recreate the input filter object.

SOURCE CODE ^

```
function cmd = string(f, varargin)
% STRING writes a command string that can be used to recreate the input filter object.
%
%%%
```

```

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ao_fcn_template

PURPOSE ^

FOO overloads the foo function for Analysis Objects.

SYNOPSIS ^

function varargout = foo(varargin)

DESCRIPTION ^

```
 FOO overloads the foo function for Analysis Objects.

%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = foo(varargin)
% FOO overloads the foo function for Analysis Objects.
%
%%%
```

```
VERSION  = '$Id: ao_fcn_template.m,v 1.7 2008/01/15 06:24:51 mauro Exp $';
bs       = [];

%% Check if this is a call for parameters or the cvs version string
if nargin == 2
  if 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
    end
  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
N = find(pl, 'N');
X = find(pl, 'X');

%% 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 = y;
  x = 1:length(y);

  %%-------- Build output AOs

  % make a new xydata object, for example
  xy = xydata(x, y);
  xy = set(xy, 'name', sprintf('foo(%s)', a.data.name));
```

```
  xy = set(xy, 'xunits', a.data.yunits);
  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 for this object
  b  = setnh(b, 'name', sprintf('foo(%s)', invars{j}));
  bs = [bs b];

end

%% Reshape the ouput to the same size of the input
varargout{1} = reshape(bs, size(as));

%% %%%%
```

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(history, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(history, 'Version')

              The following call returns the category:
              >> cat = isfield(history, 'Category')

 VERSION:     $Id: isfield.m,v 1.4 2008/02/15 16:27:15 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)

% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
  if ischar(varargin{j})
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 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 1.5 2007/10/24 10:55:11 ingo Exp $

 HISTORY: 27-08-2002 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function f = mkhighpass(f, pl)
% 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 1.5 2007/10/24 10:55:11 ingo Exp $
%
% HISTORY: 27-08-2002 M Hewitson
%             Creation
%
%%%
```

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input param object.

SYNOPSIS ^

function cmd = string(objs, varargin)

DESCRIPTION ^

```
 STRING writes a command string that can be used to recreate the input param object.

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

 DESCRIPTION: STRING writes a command string that can be used to recreate the
              input param object.

 CALL:        cmd = string(param_obj)

 INPUT:       param_obj - parameter object

 OUTPUT:      cmd       - command string to create the input object

 VERSION:     $Id: string.m,v 1.1 2008/02/15 17:38:59 ingo Exp $

 HISTORY:     15-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- string STRING writes a command string that can be used to recreate the input param object.

This function is called by:

- string STRING writes a command string that can be used to recreate the input param object.

SOURCE CODE ^

```
function cmd = string(objs, varargin)
% STRING writes a command string that can be used to recreate the input param object.
%
%%%
```

```
  key = obj.key;
  val = obj.val;

  if ischar(val)
    val_str = [''''' val '''''];
  elseif isnumeric(val)
    val_str = ltpda_mat2str(val);
  elseif isobject(val)
    val_str = string(val);
  else
    error('### Unknown object [%s]', class(val));
  end

  cmd = [cmd 'param(''' key ''', ' val_str ') '];
end

if length(objs) > 1
  cmd = [cmd ']'];
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

aooperate

PURPOSE ^

AOOPERATE implements specified operator overload for AOs.

SYNOPSIS ^

function [a1,a2,do] = aooperate(varargin)

DESCRIPTION ^

```
AOOPERATE implements specified operator overload for AOs.

M Hewitson 16-03-07

$Id: aooperate.m,v 1.18 2008/02/13 18:05:44 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function [a1,a2,do] = aooperate(varargin)

% AOOPERATE implements specified operator overload for AOs.
%
% M Hewitson 16-03-07
%
% $Id: aooperate.m,v 1.18 2008/02/13 18:05:44 hewitson Exp $
%

ALGONAME = mfilename;
VERSION  = '$Id: aooperate.m,v 1.18 2008/02/13 18:05:44 hewitson Exp $';

% should be possible to operate with
%     cdata and tsdata - only need to be same length if length(cdata)>1
% or  cdata and fsdata - only need to be same length if length(cdata)>1
% or  cdata and cdata
%
% ** how do we deal with units for these different types?
%  - only compare units when types are the same
```

```
op       = varargin{end};
varargin = varargin{1:end-1};

% % Extract AOs
% aos = [];
% pls = [];
% for j=1:length(varargin)
%   if isa(varargin{j}, 'ao')
%     aos = [aos varargin{j}];
%   elseif isa(varargin{j}, 'plist')
%     pls = [pls varargin{j}];
%   else
%   end
% end

nargin   = length(varargin);

as = [];
for j=1:nargin
  a = varargin{j};
  if isa(a, 'ao')
    if any(size(a) > 1)
      error('### The operation works only with single analysis objects. Do not use vector or matrices of analysis objects.')
    end
    as  = [as a];
  elseif isnumeric(a)
    % create an ao from the values
    as = [as ao(a)];
  elseif isa(a, 'plist')
    val = find(a, 'Value');
    if ~isempty(val)
      as = [as ao(val)];
    end
  else
    error('### AO/plus inputs must be either numeric, plist(''Value''), or AO.');
  end
end

if length(as) > 2
  warning('!!! I only work with the first two input AOs. Ignoring all others.');
  warning('!!! Perhaps you have input an AO array?');
end

%%%%%%%%%%   Now we have two AOs   %%%%%%%%%%
a1 = as(1);
a2 = as(2);
clear as;
d1 = a1.data;
d2 = a2.data;

```

```
%%%%%%%%%%   Check types   %%%%%%%%%%
if (isa(d1, 'fsdata') && isa(d2, 'tsdata')) || (isa(d1, 'tsdata') && isa(d2, 'fsdata'))
  error('### can''t add time-series data to frequency-series data.');
end
if (isa(d1, 'xydata') && isa(d2, 'tsdata')) || (isa(d1, 'tsdata') && isa(d2, 'xydata'))
  error('### can''t add XY data to time-series data.');
end
if (isa(d1, 'xydata') && isa(d2, 'fsdata')) || (isa(d1, 'fsdata') && isa(d2, 'xydata'))
  error('### can''t add XY data to frequency-series data.');
end

%%%%%%%%%%   get data   %%%%%%%%%%
[x1, y1] = get_xy_values(a1.data);
[x2, y2] = get_xy_values(a2.data);
ny1 = length(y1);
ny2 = length(y2);

% Some special cases
% switch op
%   case '*'
%     if size(y1) == size(y2)
%       y1 = y1.';
%     end
% end


%%%%%%%%%%   check data lengths   %%%%%%%%%%
if isa(d1, 'cdata') && ~isa(d2, 'cdata')
  if ny1 > 1 && ny1~=ny2
    error('###1 cdata type must be same length as other AO, or single valued.');
  end
elseif isa(d2, 'cdata') && ~isa(d1, 'cdata')
  if ny2 > 1 && ny1~=ny2
    error('###2 cdata type must be same length as other AO, or single valued.');
  end
end

if ny1~=ny2
  if ny1~=1 && ny2~=1
    error('### data vectors must be the same length or one should be single valued.');
  end
end

%%%%%%%%%%   check units   %%%%%%%%%%
if isa(d1, 'fsdata') && isa(d2, 'fsdata')
  if ~strcmp(d1.xunits, d2.xunits)
    error('### xunits must be the same for same type of AOs.');
  end
```

```
  % check X-axis
  if d1.x ~= d2.x
    error('### frequency vectors are not the same.');
  end
end
if isa(d1, 'tsdata') && isa(d2, 'tsdata')
  if ~strcmp(d1.xunits, d2.xunits)
    error('### xunits must be the same for same type of AOs.');
  end
end
if isa(d1, 'xydata') && isa(d2, 'xydata')
  if ~strcmp(d1.xunits, d2.xunits)
    error('### xunits must be the same for same type of AOs.');
  end
end

%%%%%%%%%%   Add the data   %%%%%%%%%%
% disp(sprintf('*** applying %s', op))
eval(sprintf('y = y1%sy2;', op));

%%%%%%%%%%   Create output data object   %%%%%%%%%%

if isa(d1, 'fsdata') || isa(d2, 'fsdata')
  % then we create an output fsdata
  % select which object gives properties to
  % the output object
  if isa(d1, 'fsdata')
    d = d1;
  else
    d = d2;
  end
  % then we create an output tsdata
  do = fsdata(d.x, y, d.fs);
  do = set(do, 'name', sprintf('(%s) %s (%s)', d1.name, op, d2.name));
  do = set(do, 'xunits', d.xunits);
  do = set(do, 'yunits', d.yunits);
  do = set(do, 'enbw', d.enbw);

elseif isa(d1, 'tsdata') || isa(d2, 'tsdata')
  % select which object gives properties to
  % the output object
  if isa(d1, 'tsdata')
    d = d1;
  else
    d = d2;
  end
  % then we create an output tsdata
  do = d;
  do = set(do, 'y', y);
  do = set(do, 'name', sprintf('(%s) %s (%s)', d1.name, op, d2.name));

elseif isa(d1, 'xydata') || isa(d2, 'xydata')
  % select which object gives properties to
```

```
  % the output object
  if isa(d1, 'xydata')
    d = d1;
  else
    d = d2;
  end
  % then we create an output xydata
  do = xydata(d.x, y);
  do = set(do, 'name', sprintf('(%s) %s (%s)', d1.name, op, d2.name));
  do = set(do, 'xunits', d.xunits);
  do = set(do, 'yunits', d.yunits);

elseif isa(d1, 'cdata') || isa(d2, 'cdata')
  % then we create an output cdata
  % select which object gives properties to
  % the output object
  if isa(d1, 'cdata')
    d = d1;
  else
    d = d2;
  end
  % then we create an output cdata
  do = cdata(y);
  do = set(do, 'name', sprintf('(%s) %s (%s)', d1.name, op, d2.name));
  do = set(do, 'xunits', d.xunits);
  do = set(do, 'yunits', d.yunits);

else
  error('### unknown data type - 2');
end


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(mfir, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(mfir, 'Version')

              The following call returns the category:
              >> cat = isfield(mfir, 'Category')

 VERSION:     $Id: isfield.m,v 1.4 2008/02/15 16:27:16 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)

% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
  if ischar(varargin{j})
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 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().

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

 DESCRIPTION: MKLOWPASS return a low pass filter mfir().

 CALL:        f = mklowpass(f, pl)

 VERSION: $Id: mklowpass.m,v 1.4 2007/10/24 10:55:11 ingo Exp $

 HISTORY: 27-08-2002 M Hewitson
            Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function f = mklowpass(f, pl)
% MKLOWPASS return a low pass filter mfir().
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: MKLOWPASS return a low pass filter mfir().
%
% CALL:        f = mklowpass(f, pl)
%
```

```
% VERSION: $Id: mklowpass.m,v 1.4 2007/10/24 10:55:11 ingo Exp $
%
% HISTORY: 27-08-2002 M Hewitson
%             Creation
%
%%%
```

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(miir, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(miir, 'Version')

              The following call returns the category:
              >> cat = isfield(miir, 'Category')

 VERSION:     $Id: isfield.m,v 1.4 2008/02/15 16:27:16 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)

% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
  if ischar(varargin{j})
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 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 1.5 2007/10/24 10:55:11 ingo Exp $

 HISTORY: 27-08-2002 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function f = mklowpass(f, pl)
% 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 1.5 2007/10/24 10:55:11 ingo Exp $
%
% HISTORY: 27-08-2002 M Hewitson
%             Creation
%
%%%
```

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input pole object.

SYNOPSIS ^

function cmd = string(ps, varargin)

DESCRIPTION ^

```
 STRING writes a command string that can be used to recreate the input pole object.

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

 DESCRIPTION: STRING writes a command string that can be used to recreate the
              input pole object.

 CALL:        cmd = string(pole);

 VERSION:     $Id: string.m,v 1.7 2008/02/15 17:38:59 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 pole object.

This function is called by:

- string STRING writes a command string that can be used to recreate the input pole object.

SOURCE CODE ^

```
function cmd = string(ps, varargin)
% STRING writes a command string that can be used to recreate the input pole object.
%
```

```
%%%
```

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(param, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(param, 'Version')

              The following call returns the category:
              >> category = isfield(param, 'Category')

 VERSION:     $Id: isfield.m,v 1.3 2008/02/13 17:33:38 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)

% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
  if ischar(varargin{j})
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input provenance object.

SYNOPSIS ^

function cmd = string(objs, 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

 VERSION:     $Id: string.m,v 1.1 2008/02/15 17:38:59 ingo Exp $

 HISTORY:     15-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a provenance object into a string.

This function is called by:

SOURCE CODE ^

```
function cmd = string(objs, varargin)
```

```
% STRING writes a command string that can be used to recreate the input provenance object.
%
%%%
```

```

  cmd = [cmd 'provenance(''' creator ''') '];
end

if length(objs) > 1
  cmd = [cmd ']'];
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input pzmodel object.

SYNOPSIS ^

function cmd = string(p, varargin)

DESCRIPTION ^

```
 STRING writes a command string that can be used to recreate the input pzmodel
object.

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

 DESCRIPTION: STRING writes a command string that can be used to recreate
              the input pzmodel object.

 CALL:        cmd = string(pzmodel)

 VERSION:     $Id: string.m,v 1.7 2008/02/23 19:49:11 hewitson Exp $

 HISTORY:     02-04-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pzmodel object into a string.
- string STRING writes a command string that can be used to recreate the input pzmodel object.

This function is called by:

- string STRING writes a command string that can be used to recreate the input pzmodel object.

SOURCE CODE ^

```
function cmd = string(p, varargin)
% STRING writes a command string that can be used to recreate the input pzmodel
```

```
object.
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(pole, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(pole, 'Version')

              The following call returns the category:
              >> category = isfield(pole, 'Category')

 VERSION:     $Id: isfield.m,v 1.4 2008/02/15 16:27:16 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)

% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
  if ischar(varargin{j})
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input window object.

SYNOPSIS ^

function cmd = string(sw ,varargin)

DESCRIPTION ^

```
 STRING writes a command string that can be used to recreate the input window object.

%%%
```

```
%
%%%
```

```
        cmd = [cmd ' specwin(''' s.name ''', ' num2str(length(s.win)) ')'];
    end
  end
end

if length(sw) > 1
  cmd = [cmd ']'];
end


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(provenance, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(provenance, 'Version')

              The following call returns the category:
              >> category = isfield(provenance, 'Category')

 VERSION:     $Id: isfield.m,v 1.3 2008/02/13 17:11:03 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)

% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
  if ischar(varargin{j})
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 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:

- sqlResultsGUI SQLRESULTSGUI shows the results of a query in a new window.

SUBFUNCTIONS ^

- function componentDelete(src, evd)

SOURCE CODE ^

```
0001 function [table, container] = muitable(varargin)
0002
0003 % UITABLE creates a two dimensional graphic uitable component in a figure
window.
```

```
%     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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input time object.

SYNOPSIS ^

function cmd = string(tt, 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(pole);

 VERSION:     $Id: string.m,v 1.4 2008/02/19 12:10:23 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:

- format FORMAT Returns the time in the handover format.
- string STRING writes a command string that can be used to recreate the input time object.

SOURCE CODE ^

```
function cmd = string(tt, varargin)
% STRING writes a command string that can be used to recreate the input time
```

```
object.
%
%%%
```

```
0054
0055 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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
  pmh = findobj('Tag', sprintf('c%02dField', k));
  conds{k,1} = get(pmh, 'Value');
  pmh = findobj('Tag', sprintf('c%02dCondition', k));
  conds{k,2} = get(pmh, 'Value');
  pmh = findobj('Tag', sprintf('c%02dValue', k));
  conds{k,3} = get(pmh, 'String');
  if k<Nc
    pmh = findobj('Tag', sprintf('c%02dCompound', k));
    conds{k,4} = get(pmh, 'Value');
  end
end

delete(get(panel, 'Children'))
drawnow
%-----------------------------------------------------------------------
%-- Draw Table selection

% get tables
tables  = getTables(conn);
Sleft   = 10;
Sheight = ALLHEIGHT;
Sbottom = Ppos(2) + Ppos(4) - 20 - Sheight;
Swidth  = 50;

sth = uicontrol(panel,'Style','text',...
                'String', 'From',...
                'Fontsize', fontsize,...
                'BackgroundColor', Gproperties.Gcol,...
```

```
                'HorizontalAlignment', 'left',...
                'Position',[Sleft Sbottom Swidth Sheight]);

Sleft  = Sleft + Swidth + 10;
Swidth = 150;
pmh = uicontrol(panel,'Style','popupmenu',...
                'String', tables,...
                'Fontsize', fontsize,...
                'BackgroundColor', Gproperties.Gcol,...
                'Value', 1,...
                'Position',[Sleft Sbottom Swidth Sheight],...
                'Tag', 'tableSelect');
if ~isempty(tableSelected)
  set(pmh, 'Value', tableSelected);
end
set(pmh, 'Callback', {'cb_tableSelect', pmh, mainfig});

%-----------------------------------------------------------------------
%-- Draw Fields Selection list

% get fields
fields = getFields(conn);
pos = get(pmh, 'Position');
Sleft   = 10;
Sheight = ALLHEIGHT;
Sbottom = pos(2) + pos(4) - 40 - Sheight;
Swidth  = 50;

sth = uicontrol(panel,'Style','text',...
                'String', 'Select',...
                'Fontsize', fontsize,...
                'BackgroundColor', Gproperties.Gcol,...
                'HorizontalAlignment', 'left',...
                'Position',[Sleft Sbottom Swidth Sheight]);

Sleft   = Sleft + Swidth + 10;
Swidth  = 150;
Sheight = 200;
Sbottom = pos(2) + pos(4) - 40 - Sheight;
lbh = uicontrol(panel,'Style','listbox',...
                'String', fields,...
                'Value',1,...
                'BackgroundColor', 'w',...
                'Fontsize', fontsize,...
                'Max', 1000,...
                'Position',[Sleft Sbottom Swidth Sheight],...
                'Tag', 'fieldsList');

set(lbh, 'Callback', {'cb_fieldSelect', pmh, mainfig});
if ~isempty(fselection)
```

```
  set(lbh, 'Value', fselection);
end

%-----------------------------------------------------------------------
%-- Query display
listpos = get(lbh, 'Position');
Sleft   = 10;
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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(pzmodel, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(pzmodel, 'Version')

              The following call returns the category:
              >> category = isfield(pzmodel, 'Category')

 VERSION:     $Id: isfield.m,v 1.4 2008/02/15 16:27:16 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)
% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input timeformat object.

SYNOPSIS ^

function cmd = string(objs, varargin)

DESCRIPTION ^

```
 STRING writes a command string that can be used to recreate the input timeformat
object.

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

 DESCRIPTION: STRING writes a command string that can be used to recreate the
              input timeformat object.

 CALL:        cmd = string(obj)

 INPUT:       obj - timeformat object

 OUTPUT:      cmd - command string to create the input object

 VERSION:     $Id: string.m,v 1.1 2008/02/15 17:38:59 ingo Exp $

 HISTORY:     15-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a timeformat object into a string.

This function is called by:

SOURCE CODE ^

```
0001 function cmd = string(objs, varargin)
```

```
% STRING writes a command string that can be used to recreate the input timeformat object.
%
%%%
```

```
0053   default_matlab_str = obj.default_matlab_str;
0054   default_matlab_nr  = obj.default_matlab_nr;
0055   default_java_str   = obj.default_java_str;
0056
0057   cmd = [cmd 'timeformat(''' format_str ''', ''' default_matlab_str ''', '
num2str(default_matlab_nr) ', ''' default_java_str ''') '];
0058 end
0059
0060 if length(objs) > 1
0061   cmd = [cmd ']'];
0062 end
0063
0064 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(specwin, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(specwin, 'Version')

              The following call returns the function category:
              >> category = isfield(specwin, 'Category')

 VERSION:     $Id: isfield.m,v 1.3 2008/02/13 13:46:07 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)
% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
  if ischar(varargin{j})
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

mux

PURPOSE ^

MUX concatenate params into a vector.

SYNOPSIS ^

function b = mux(varargin)

DESCRIPTION ^

```
 MUX concatenate params into a vector.

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

 DESCRIPTION: MUX concatonate params into a vector.

 CALL:        param_vec = mux(p1, p2);

 VERSION:     $Id: mux.m,v 1.4 2008/02/13 17:33:38 mauro Exp $

 HISTORY:     27-03-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SOURCE CODE ^

```
function b = mux(varargin)
% MUX concatenate params into a vector.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: MUX concatonate params into a vector.
%
```

```
% CALL:        param_vec = mux(p1, p2);
%
% VERSION:     $Id: mux.m,v 1.4 2008/02/13 17:33:38 mauro Exp $
%
% HISTORY:     27-03-2007 M Hewitson
%                 Creation
%
%%%
```

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input timespan object.

SYNOPSIS ^

function cmd = string(objs, 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

 VERSION:     $Id: string.m,v 1.2 2008/02/23 19:49:11 hewitson Exp $

 HISTORY:     15-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a timespan object into a string.
- string STRING writes a command string that can be used to recreate the input timespan object.

This function is called by:

- string STRING writes a command string that can be used to recreate the input timespan object.

SOURCE CODE ^

```
function cmd = string(objs, varargin)
% STRING writes a command string that can be used to recreate the input timespan object.
%
%%%
```

```

for j=1:length(objs)
  obj = objs(j);
  if ~isempty(obj.plist)
    cmd = [cmd sprintf('timespan(%s)', string(obj.plist))];
  else
    ts_start    = obj.start;
    ts_end      = obj.end;
    ts_timezone = obj.timezone;
    ts_format   = obj.timeformat;

    %% Convert the java object into a string
    ts_timezone = char(ts_timezone.getID);

    cmd = [cmd 'timespan( plist(''start'', '''       char(ts_start) ''', ' ...
      '''end'', '''         char(ts_end)   ''', ' ...
      '''timezone'', '''       ts_timezone ''', ' ...
      '''timeformat'', ''' char(ts_format) ''') ) '];
  end
end

if length(objs) > 1
  cmd = [cmd ']'];
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(time, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(time, 'Version')

              The following call returns the category:
              >> category = isfield(time, 'Category')

 VERSION:     $Id: isfield.m,v 1.4 2008/02/13 16:47:35 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)
% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

mysql_connect

PURPOSE ^

MYSQL_CONNECT connects to an AO repository and returns the connection object.

SYNOPSIS ^

function [conn] = mysql_connect(varargin)

DESCRIPTION ^

```
 MYSQL_CONNECT connects to an AO repository and returns the connection object.

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

 DESCRIPTION: MYSQL_CONNECT connects to an AO repository and returns the
              connection object.

 CALL:        conn = mysql_connect(hostname, dbname)

 VERSION:     $Id: mysql_connect.m,v 1.10 2008/02/24 10:16:58 hewitson Exp $

 HISTORY:     24-05-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- logindlg LOGINDLG Dialog for visually secure login.

This function is called by:

- ltpda_getAOsInTimeSpan LTPDA_GETAOSINTIMESPAN performs high-level queries to retrieve AOs from an LTPDA repository.

SOURCE CODE ^

```
function [conn] = mysql_connect(varargin)
% MYSQL_CONNECT connects to an AO repository and returns the connection object.
%
```

```
%%%
```

```
disp(ping(conn))


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input zero object.

SYNOPSIS ^

function cmd = string(ps, varargin)

DESCRIPTION ^

```
 STRING writes a command string that can be used to recreate the input zero object.

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

 DESCRIPTION: STRING writes a command string that can be used to recreate the
              input zero object.

 CALL:        cmd = string (zero);

 VERSION:     $Id: string.m,v 1.7 2008/02/15 17:38:59 ingo Exp $

 HISTORY:     02-04-2007 Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- string STRING writes a command string that can be used to recreate the input zero object.

This function is called by:

- string STRING writes a command string that can be used to recreate the input zero object.

SOURCE CODE ^

```
function cmd = string(ps, varargin)
% STRING writes a command string that can be used to recreate the input zero object.
%
```

```
%%%
```

cat

PURPOSE ^

CAT concatonate params into a vector.

SYNOPSIS ^

function b = cat(varargin)

DESCRIPTION ^

```
 CAT concatonate params into a vector.

%%%
```

```
% CALL:        p_vector = cat(p1, p2, p3);
%
% VERSION:     $Id: cat.m,v 1.4 2008/02/13 17:33:07 mauro Exp $
%
% HISTORY:     27-03-2007 M Hewitson
%                 Creation
%
%%%
```

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(timeformat, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(timeformat, 'Version')

              The following call returns the category:
              >> category = isfield(timeformat, 'Category')

 VERSION:     $Id: isfield.m,v 1.3 2008/02/13 16:54:14 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)
% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

mysql_getMD5hash

PURPOSE ^

MYSQL_GETMD5HASH gets the md5 hash string of a given object ID.

SYNOPSIS ^

function h = mysql_getMD5hash(conn, id)

DESCRIPTION ^

```
 MYSQL_GETMD5HASH gets the md5 hash string of a given object ID.

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

 DESCRIPTION: MYSQL_GETMD5HASH gets the md5 hash string of a given object ID.

 CALL:        h = mysql_getMD5hash(conn, id)

 VERSION:     $Id: mysql_getMD5hash.m,v 1.1 2007/09/15 04:54:44 hewitson Exp $

 HISTORY:     24-05-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function h = mysql_getMD5hash(conn, id)

% MYSQL_GETMD5HASH gets the md5 hash string of a given object ID.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: MYSQL_GETMD5HASH gets the md5 hash string of a given object ID.
%
% CALL:        h = mysql_getMD5hash(conn, id)
```

```
%
% VERSION:     $Id: mysql_getMD5hash.m,v 1.1 2007/09/15 04:54:44 hewitson Exp $
%
% HISTORY:     24-05-2007 M Hewitson
%                 Creation
%
%%%
```

strpad

PURPOSE ^

STRPAD Pads a string with blank spaces until it is N characters long.

SYNOPSIS ^

function so = strpad(s, N)

DESCRIPTION ^

```
 STRPAD Pads a string with blank spaces until it is N characters long.

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

 DESCRIPTION: STRPAD Pads a string with blank spaces until it is N characters
              long. If s is already N characters long (or longer) then the
              string is truncated.

 CALL:        so = strpad('Pads this string to 30 characters', 30)

 INPUTS:      s  - string
              N  - length of the string

 OUTPUTS:     so - the padded string

 VERSION: $Id: strpad.m,v 1.2 2007/07/13 12:17:39 ingo Exp $

 HISTORY: 27-04-2007 M Hewitson
              Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function so = strpad(s, N)
% STRPAD Pads a string with blank spaces until it is N characters long.
```

```
%
%%%
```

cat

PURPOSE ^

CAT concatonate poles into a vector.

SYNOPSIS ^

function b = cat(varargin)

DESCRIPTION ^

```
 CAT concatonate poles into a vector.

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

 DESCRIPTION: CAT concatonate poles into a vector.

 CALL:        pole_vector = cat(pole1, pole2, pole3);

 VERSION:     $Id: cat.m,v 1.4 2008/02/12 21:38:23 hewitson Exp $

 HISTORY:     27-03-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pole object into a string.

This function is called by:

SOURCE CODE ^

```
function b = cat(varargin)
% CAT concatonate poles into a vector.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: CAT concatonate poles into a vector.
%
```

```
% CALL:        pole_vector = cat(pole1, pole2, pole3);
%
% VERSION:     $Id: cat.m,v 1.4 2008/02/12 21:38:23 hewitson Exp $
%
% HISTORY:     27-03-2007 M Hewitson
%                 Creation
%
%%%
```

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(timespan, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(timespan, 'Version')

              The following call returns the category:
              >> category = isfield(timespan, 'Category')

 VERSION:     $Id: isfield.m,v 1.3 2008/02/13 16:43:14 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)
% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

mysql_getMaxId

PURPOSE ^

MYSQL_GETMAXID get the maximum Id from the objs table.

SYNOPSIS ^

function id = mysql_getMaxId(conn)

DESCRIPTION ^

```
 MYSQL_GETMAXID get the maximum Id from the objs table.

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

 DESCRIPTION: MYSQL_GETMAXID get the maximum Id from the objs table.

 CALL:        id = mysql_getMaxId()

 VERSION:     $Id: mysql_getMaxId.m,v 1.3 2007/08/31 17:29:22 hewitson Exp $

 HISTORY:     24-05-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function id = mysql_getMaxId(conn)
% MYSQL_GETMAXID get the maximum Id from the objs table.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: MYSQL_GETMAXID get the maximum Id from the objs table.
%
% CALL:        id = mysql_getMaxId()
%
```

```
% VERSION:     $Id: mysql_getMaxId.m,v 1.3 2007/08/31 17:29:22 hewitson Exp $
%
% HISTORY:     24-05-2007 M Hewitson
%                 Creation
%
%%%
```

cat

PURPOSE ^

CAT concatonate zeros into a vector.

SYNOPSIS ^

function b = cat(varargin)

DESCRIPTION ^

```
 CAT concatonate zeros into a vector.

%%%
```

```
% CALL:        zero_vector = cat(zero1, zero2, zero3)
%
% VERSION:     $Id: cat.m,v 1.4 2008/02/12 19:54:16 hewitson Exp $
%
% HISTORY:     27-03-2007 Hewitson
%                 Creation
%
%%%
```

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(tsdata, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(tsdata, 'Version')

              The following call returns the category:
              >> category = isfield(ao, 'Category')

 VERSION:     $Id: isfield.m,v 1.4 2008/02/15 16:27:16 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)
% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
    as = [as varargin{j}];
  end
  if ischar(varargin{j})
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

mysql_getObjIds

PURPOSE ^

MYSQL_GETOBJIDS gets a list of object ids from a given collection id.

SYNOPSIS ^

function ids = mysql_getObjIds(conn, cid)

DESCRIPTION ^

```
MYSQL_GETOBJIDS gets a list of object ids from a given collection id.

Usage: ids = mysql_getObjIds(conn,cid);

Inputs:
    conn - a database connection object
    cid  - a collection id

Outputs:
    ids  - a list of object ids

M Hewitson 30-08-07

$Id: mysql_getObjIds.m,v 1.3 2008/02/22 09:25:18 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpda_obj_retrieve LTPDA_OBJ_RETRIEVE retrieves a collection of objects from an LTPDA repository.

SOURCE CODE ^

```
function ids = mysql_getObjIds(conn, cid)

% MYSQL_GETOBJIDS gets a list of object ids from a given collection id.
%
% Usage: ids = mysql_getObjIds(conn,cid);
%
% Inputs:
```

```
%     conn - a database connection object
%     cid  - a collection id
%
% Outputs:
%     ids  - a list of object ids
%
% M Hewitson 30-08-07
%
% $Id: mysql_getObjIds.m,v 1.3 2008/02/22 09:25:18 hewitson Exp $
%

try
  curs    = exec(conn, sprintf('select obj_ids from collections where id="%d"', cid));
  curs    = fetch(curs);
  ids     = str2num(curs.Data{1});
  close(curs);
catch
  error('### Unable to determine object IDs from collection ID. Server returned %s', curs.Message);
end


```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsasgn

PURPOSE ^

SUBSASGN define index assignment for fsdata properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for fsdata properties.

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

 DESCRIPTION: SUBSASGN define index assignment for fsdata
              properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.7 2008/02/25 18:06:31 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

 $Id: subsasgn.m,v 1.7 2008/02/25 18:06:31 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

- cdata CDATA constant data object class constructor.
- char CHAR convert a cdata object into a string.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for fsdata properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
% DESCRIPTION: SUBSASGN define index assignment for fsdata
%              properties.
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.7 2008/02/25 18:06:31 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
% $Id: subsasgn.m,v 1.7 2008/02/25 18:06:31 ingo Exp $


% Check if this is a call for parameters
if nargin == 2
  if isa(A, 'cdata') && ischar(S)
    in = char(S);
    if strcmp(in, 'Params')
      A = plist;
      return
    elseif strcmp(in, 'Version')
      VERSION  = '$Id: subsasgn.m,v 1.7 2008/02/25 18:06:31 ingo Exp $';
      A = VERSION;
      return
    elseif strcmp(in, 'Category')
      CATEGORY = 'Internal';
      ts = CATEGORY;
      return
    end
  end
end

if isempty(A)
  A = cdata();
end

A = builtin ('subsasgn', A, S, B);
```

Generated on Tue 26-Feb-2008 10:52:52 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};
```

```
0012
0013 id = findobj('Tag', 'methodInfoPanel');
0014
0015 % get selected method
0016 id = findobj('Tag', 'classMethodList');
0017 ostr = get(id, 'String');
0018 oidx = get(id, 'Value');
0019
0020 objStr = ostr(oidx);
0021
0022 id = findobj('Tag', 'methodInfoPanel');
0023
0024 if length(oidx) == 1
0025   s = help(char(objStr));
0026   set(id, 'String', s);
0027 end
0028
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(xydata, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(xydata, 'Version')

              The following call returns the routine category:
              >> category = isfield(xydata, 'Category')

 VERSION:     $Id: isfield.m,v 1.4 2008/02/15 16:27:16 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)
% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
    as = [as varargin{j}];
  end
  if ischar(varargin{j})
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

mysql_getObjType

PURPOSE ^

MYSQL_GETOBJTYPE gets the object type associated with the given object id.

SYNOPSIS ^

function type = mysql_getObjType(conn, id)

DESCRIPTION ^

```
MYSQL_GETOBJTYPE gets the object type associated with the given object id.

Usage: ids = mysql_getObjType(conn,id);

Inputs:
    conn - a database connection object
    id   - an object id

Outputs:
    type - the type of this object

M Hewitson 30-08-07

$Id: mysql_getObjType.m,v 1.3 2008/02/22 09:25:18 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function type = mysql_getObjType(conn, id)

% MYSQL_GETOBJTYPE gets the object type associated with the given object id.
%
% Usage: ids = mysql_getObjType(conn,id);
%
% Inputs:
%     conn - a database connection object
%     id   - an object id
```

```
%
% Outputs:
%     type - the type of this object
%
% M Hewitson 30-08-07
%
% $Id: mysql_getObjType.m,v 1.3 2008/02/22 09:25:18 hewitson Exp $
%


try
  curs    = exec(conn, sprintf('select obj_type from objmeta where obj_id="%d"', id));
  curs    = fetch(curs);
  type    = curs.Data{1};
  close(curs);
catch
  warning('!!! Unable to get object type for ID %d. [Server returned: %s]', id, curs.Message);
  type = '';
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsasgn

PURPOSE ^

SUBSASGN define index assignment for fsdata properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for fsdata properties.

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

 DESCRIPTION: SUBSASGN define index assignment for fsdata
              properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.6 2008/02/25 18:06:31 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- fsdata FSDATA frequency-series object class constructor.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for fsdata properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
% DESCRIPTION: SUBSASGN define index assignment for fsdata
%               properties.
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.6 2008/02/25 18:06:31 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(xyzdata, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(xyzdata, 'Version')

              The following call returns the routine category:
              >> category = isfield(xyzdata, 'Category')

 VERSION:     $Id: isfield.m,v 1.3 2008/02/15 16:27:16 mauro Exp $

 HISTORY: 24-12-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)
% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

mysql_getUserID

PURPOSE ^

MYSQL_GETUSERID gets the user ID number corresponding to the given user name.

SYNOPSIS ^

function [userid,dbuser] = mysql_getUserID(conn, username)

DESCRIPTION ^

```
 MYSQL_GETUSERID gets the user ID number corresponding to the given user name.

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

 DESCRIPTION: MYSQL_GETUSERID gets the user ID number corresponding to the
              user name that connected to the MySQL database.

 CALL:        userid = mysql_getUserID(conn)

 VERSION:     $Id: mysql_getUserID.m,v 1.7 2008/02/24 10:16:58 hewitson Exp $

 HISTORY:     24-05-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpda_obj_retrieve LTPDA_OBJ_RETRIEVE retrieves a collection of objects from an LTPDA repository.
- ltpda_obj_submit LTPDA_OBJ_SUBMIT submits the given collection of objects to an LTPDA Repository.

SOURCE CODE ^

```
function [userid,dbuser] = mysql_getUserID(conn, username)
% MYSQL_GETUSERID gets the user ID number corresponding to the given user name.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
% DESCRIPTION: MYSQL_GETUSERID gets the user ID number corresponding to the
%               user name that connected to the MySQL database.
%
% CALL:         userid = mysql_getUserID(conn)
%
% VERSION:      $Id: mysql_getUserID.m,v 1.7 2008/02/24 10:16:58 hewitson Exp $
%
% HISTORY:      24-05-2007 M Hewitson
%                  Creation
%
%%%
```

subsasgn

PURPOSE ^

SUBSASGN define index assignment for history properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for history properties.

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

 DESCRIPTION: SUBSASGN define index assignment for history
              properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.5 2008/02/25 18:06:31 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- history HISTORY History object class constructor.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for history properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
% DESCRIPTION: SUBSASGN define index assignment for history
%              properties.
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.5 2008/02/25 18:06:31 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isfield

PURPOSE ^

ISFIELD tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isfield(varargin)

DESCRIPTION ^

```
 ISFIELD tests if the given field is one of the object properties.

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

 DESCRIPTION: ISFIELD tests if the given field is one of the object
              properties.

 CALL:        isfield(a, 'field');

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isfield(zero, 'Params')

              The following call returns the cvs version number:
              >> ver = isfield(zero, 'Version')

              The following call returns the category:
              >> category = isfield(zero, 'Category')

 VERSION:     $Id: isfield.m,v 1.4 2008/02/15 16:27:16 mauro Exp $

 HISTORY: 25-09-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isfield(varargin)

% ISFIELD tests if the given field is one of the object properties.
%
%%%
```

```
  if ischar(varargin{j})
    field = varargin{j};
  end
end

res = zeros(size(as));
for j=1:numel(as)
  try
    t = as(j).(field);
    res(j) = 1;
  catch
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

mysql_getXdoc

PURPOSE ^

MYSQL_GETXDOC retrieves an object with given id from the LTPDA

SYNOPSIS ^

function xdoc = mysql_getXdoc(conn, id)

DESCRIPTION ^

```
MYSQL_GETXDOC retrieves an object with given id from the LTPDA
repository specified by the input database connection. The object is
converted from its XML text format to an Xdoc. This can then be converted
into an object using the appropriate object constructor.

Usage: xdoc = mysql_getXdoc(conn, id)

Inputs:
        conn - a database connection object
        id   - the object id

Outputs:
        xdoc - an Xdoc representation of the object.

M Hewitson 30-08-07

$Id: mysql_getXdoc.m,v 1.4 2008/02/22 09:25:18 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpda_obj_retrieve LTPDA_OBJ_RETRIEVE retrieves a collection of objects from an LTPDA repository.

SOURCE CODE ^

```
function xdoc = mysql_getXdoc(conn, id)
```

```

% MYSQL_GETXDOC retrieves an object with given id from the LTPDA
% repository specified by the input database connection. The object is
% converted from its XML text format to an Xdoc. This can then be converted
% into an object using the appropriate object constructor.
%
% Usage: xdoc = mysql_getXdoc(conn, id)
%
% Inputs:
%         conn - a database connection object
%         id   - the object id
%
% Outputs:
%         xdoc - an Xdoc representation of the object.
%
% M Hewitson 30-08-07
%
% $Id: mysql_getXdoc.m,v 1.4 2008/02/22 09:25:18 hewitson Exp $
%
%


try
  curs    = exec(conn, sprintf('select xml from objs where id="%d"', id));
  curs    = fetch(curs);
  objTxt  = char([curs.Data{1}].');
  close(curs);
catch
  error('### Unable to read xml for ID %d. Server returned %s', id, curs.Message);
end

if ~isempty(objTxt) && ~strcmp(objTxt(:), 'No Data')
  % convert to Java string
  str = java.lang.String(objTxt);
  % open stream on this string
  stream = java.io.StringBufferInputStream(str);
  % make parser
  factory = javaMethod('newInstance',...
    'javax.xml.parsers.DocumentBuilderFactory');
  builder = factory.newDocumentBuilder;

  xdoc = builder.parse(stream);
else
  xdoc = [];
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsasgn

PURPOSE ^

SUBSASGN define index assignment for mfir properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for mfir properties.

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

 DESCRIPTION: SUBSASGN define index assignment for mfir properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:31 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a mfir object into a string.
- mfir MFIR FIR filter object class constructor.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for mfir properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSASGN define index assignment for mfir properties.
```

```
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:31 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

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 1.3 2008/02/24 10:17:30 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 1.3 2008/02/24 10:17:30 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');


  % connect
  conn = mysql_connect(hostname, db);
  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));

  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');
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

islinespec

PURPOSE ^

ISLINESPEC checks a string to the line spec syntax.

SYNOPSIS ^

function varargout = islinespec(str)

DESCRIPTION ^

```
 ISLINESPEC checks a string to the line spec syntax.
%%%
```

```
% DESCRIPTION: ISLINESPEC checks a string to the line spec syntax.
%
% CALL:        ret               = islinespec(str);
%              [style_array ret] = islinespec(str);
%
% REMARK:      The style_array is a cell array with the line style, marker style
%              and the color style.
%              style_array{1}: line style
%              style_array{2}: marker style
%              style_array{3}: color style
%
% VERSION:     $Id: islinespec.m,v 1.3 2007/08/24 13:05:09 ingo Exp $
%
% HISTORY:     21-08-2007 Diepholz
%                 Creation
%
%%%
```

```

%%%%%%%%%%   Check full name marker style   %%%%%%%%%%
for ii = 1:length(full_name_marker_style)
  idx = strfind(copy_str, full_name_marker_style{ii});
  if ~isempty(idx)
    copy_str = strrep(copy_str, full_name_marker_style{ii}, '');
    out_marker_style = full_name_marker_style{ii};
    found_marker_style = true;
    break;
  end
end

%%%%%%%%%%   Check full name color style   %%%%%%%%%%
for ii = 1:length(full_name_color_style)
  idx = strfind(copy_str, full_name_color_style{ii});
  if ~isempty(idx)
    copy_str = strrep(copy_str, full_name_color_style{ii}, '');
    out_color_style = full_name_color_style{ii};
    found_color_style = true;
    break;
  end
end

%%%%%%%%%%   Check full marker style   %%%%%%%%%%
if found_marker_style == false
  for ii = 1:length(marker_style)
    idx = strfind(copy_str, marker_style{ii});
    if ~isempty(idx)
      copy_str = strrep(copy_str, marker_style{ii}, '');
      out_marker_style = marker_style{ii};
      break;
    end
  end
end

%%%%%%%%%%   Check full color style   %%%%%%%%%%
if found_color_style == false
  for ii = 1:length(color_style)
    idx = strfind(copy_str, color_style{ii});
    if ~isempty(idx)
      copy_str = strrep(copy_str, color_style{ii}, '');
      out_color_style = color_style{ii};
      break;
    end
  end
end

%%%%%%%%%%   Set the output   %%%%%%%%%%
if isempty(copy_str)
  ret = true;
else
  ret = false;
end
```

```

  if nargout == 0 || nargout == 1
    varargout{1} = ret;
  elseif nargout == 2
    varargout{1} = ret;
    varargout{2} = {out_line_style, out_marker_style, out_color_style};
  else
    error('### Unknown numbers of outputs.');
  end

end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsasgn

PURPOSE ^

SUBSASGN define index assignment for miir properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for miir properties.

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

 DESCRIPTION: SUBSASGN define index assignment for miir properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:31 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a miir object into a string.
- miir MIIR IIR filter object class constructor.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for miir properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSASGN define index assignment for miir properties.
```

```
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:31 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for cdata objects.

SYNOPSIS ^

function result = ne(c1, c2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for cdata objects.

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

 DESCRIPTION: NE overloads the ~= operator for cdata objects.

 CALL:        result = ne(c1,c2)
              result = ne(c1,c2, 'property1', 'property2')
              result = ne(c1,c2, 'class/property', 'class/property')
              result = ne(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      c1,c2 - input cdata objects

 OUTPUTS:     If the two cdata objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.4 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a cdata object into a string.
- eq EQ overloads the == operator for cdata objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(c1, c2, varargin)
% NE overloads the ~= operator for cdata objects.
%
%%%
```

```
if eq(c1,c2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 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
  else
    whos('results')
    whos('fields')
    sqlResultsGUI(results, fields, q);
  end
else
  error('### Unable to execute query');
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for fsdata objects.

SYNOPSIS ^

function result = ne(c1, c2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for fsdata objects.

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

 DESCRIPTION: NE overloads the ~= operator for fsdata objects.

 CALL:        result = ne(c1,c2)
              result = ne(c1,c2, 'property1', 'property2')
              result = ne(c1,c2, 'class/property', 'class/property')
              result = ne(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      c1,c2 - input fsdata objects

 OUTPUTS:     If the two fsdata objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.4 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- eq EQ overloads the == operator for fsdata objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(c1, c2, varargin)
% NE overloads the ~= operator for fsdata objects.
%
%%%
```

```
if eq(c1,c2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 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();

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isvalid

PURPOSE ^

ISVALID tests if the given cdata has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given cdata has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given cdata has all the correct fields of the
              correct type.

 CALL:        isvalid(cdata);

              Returns 1 is cdata is a valid CDATA-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(cdata, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(cdata, 'Version')

              The following call returns the category:
              >> ver = isvalid(cdata, 'Category')

 VERSION:     $Id: isvalid.m,v 1.1 2008/02/14 13:48:33 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given cdata has all the correct fields of the correct type.
%
%%%
```

```
objs = [];
for j=1:nargin
  if isa(varargin{j}, 'cdata')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)             && ...
     isnumeric (obj.y)                && ...
     iscell    (obj.x)                && ...
     ischar    (obj.xunits)           && ...
     ischar    (obj.yunits)           && ...
     ischar    (obj.version)          && ...
     isa       (obj.created, 'time')

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for history objects.

SYNOPSIS ^

function result = ne(w1,w2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for history objects.

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

 DESCRIPTION: NE overloads the ~= operator for history objects.

 CALL:        result = ne(h1,h2)
              result = ne(h1,h2, 'property1', 'property2')
              result = ne(h1,h2, 'class/property', 'class/property')
              result = ne(h1,h2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      h1,h2 - input history objects

 OUTPUTS:     If the two history objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.4 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- eq EQ overloads the == operator for history objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(w1,w2, varargin)
% NE overloads the ~= operator for history objects.
%
%%%
```

```
if eq(w1, w2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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 1.4 2008/02/25 11:54:20 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 1.4 2008/02/25 11:54:20 hewitson Exp $
%

myh     = varargin{1};
mainfig = varargin{4};

```

```
tables = {'collections', 'objmeta', 'objs', 'transactions', 'users'};

% Get conn
conn = getappdata(mainfig, 'connection');

if isempty(conn)

  % get hostname
  hostname = get(findobj('Tag','serverTxt'), 'String');

  % connect
  conn = mysql_connect(hostname);
  if ~isa(conn, 'database')
    error('# couldn''t connect to server');
  end
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
```

```
0059
0060 end
0061 close(curs);
0062
0063 % Set the list
0064 id = findobj('Tag', 'dbSelect');
0065 set(id, 'String', ltpda_dbs, 'Value', 1);
0066
0067 % Get selected DB
0068 dbh = findobj('Tag', 'dbSelect');
0069 dbs = get(dbh, 'String');
0070 db  = dbs{1};
0071
0072 % set selected DB
0073 dbh = findobj('Tag', 'databaseTxt');
0074 set(dbh, 'String', db);
0075
0076
0077
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isvalid

PURPOSE ^

ISVALID tests if the given fsdata has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given fsdata has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given fsdata has all the correct fields of the
              correct type.

 CALL:        isvalid(fsdata);

              Returns 1 is fsdata is a valid FSDATA-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(fsdata, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(fsdata, 'Version')

              The following call returns the category:
              >> ver = isvalid(fsdata, 'Category')

 VERSION:     $Id: isvalid.m,v 1.1 2008/02/14 13:48:33 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given fsdata has all the correct fields of the correct type.
%
%%%
```

```
objs = [];
for j=1:nargin
  if isa(varargin{j}, 'fsdata')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)             && ...
     isnumeric (obj.y)                && ...
     isnumeric (obj.x)                && ...
     isnumeric (obj.enbw)             && ...
     isnumeric (obj.navs)             && ...
     isnumeric (obj.fs)               && ...
     ischar    (obj.xunits)           && ...
     ischar    (obj.yunits)           && ...
     ischar    (obj.version)          && ...
     isa       (obj.created, 'time')

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for mfir objects.

SYNOPSIS ^

function result = ne(f1, f2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for mfir objects.

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

 DESCRIPTION: NE overloads the ~= operator for mfir objects.

 CALL:        result = ne(f1,f2)
              result = ne(f1,f2, 'property1', 'property2')
              result = ne(f1,f2, 'class/property', 'class/property')
              result = ne(f1,f2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      f1,f2 - input mfir objects

 OUTPUTS:     If the two mfir objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.4 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a mfir object into a string.
- eq EQ overloads the == operator for mfir objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(f1, f2, varargin)
% NE overloads the ~= operator for mfir objects.
%
%%%
```

```
if eq(f1, f2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsasgn

PURPOSE ^

SUBSASGN define index assignment for provenance properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for provenance properties.

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

 DESCRIPTION: SUBSASGN define index assignment for provenence properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:31 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a provenance object into a string.
- provenance PROVENANCE constructors for provenance class.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for provenance properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSASGN define index assignment for provenence properties.
```

```
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:31 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

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 1.2 2008/01/22 20:45:22 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 1.2 2008/01/22 20:45:22 hewitson Exp $
%

```

```
disp('* Goodbye from the LTPDA Repository GUI *')
delete(varargin{1})
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isvalid

PURPOSE ^

ISVALID tests if the given history has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given history has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given history has all the correct fields of the
              correct type.

 CALL:        isvalid(history);

              Returns 1 is history is a valid HISTORY-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(history, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(history, 'Version')

              The following call returns the category:
              >> ver = isvalid(history, 'Category')

 VERSION:     $Id: isvalid.m,v 1.1 2008/02/14 13:48:33 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given history has all the correct fields of the correct type.
%
%%%
```

```
for j=1:nargin
  if isa(varargin{j}, 'history')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)                  && ...
     ischar    (obj.version)               && ...
       (isa (obj.plist, 'plist')     || ...
        isempty(obj.plist))                && ...
       (isa (obj.inhists, 'history') || ...
        isempty(obj.inhists))              && ...
     iscell    (obj.invars)                && ...
     isnumeric (obj.n)                     && ...
     isnumeric (obj.pn)                    && ...
     isa       (obj.created, 'time')       && ...
     ischar    (obj.consver)

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for miir objects.

SYNOPSIS ^

function result = ne(f1, f2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for miir objects.

%%%
```

This function is called by:

SOURCE CODE ^

```
function result = ne(f1, f2, varargin)
% NE overloads the ~= operator for miir objects.
%
%%%
```

```
if eq(f1, f2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 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 $
%
```

```
0009
0010
0011 % get a list of objects
0012 objs = getWorkspaceObjs();
0013
0014 % set list
0015 setWorkspaceObjsList(objs);
0016
0017
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isvalid

PURPOSE ^

ISVALID tests if the given mfir has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given mfir has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given mfir has all the correct fields of the
              correct type.

 CALL:        isvalid(mfir);

              Returns 1 is mfir is a valid MFIR-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(mfir, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(mfir, 'Version')

              The following call returns the category:
              >> ver = isvalid(mfir, 'Category')

 VERSION:     $Id: isvalid.m,v 1.2 2008/02/15 17:36:45 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given mfir has all the correct fields of the correct type.
%
%%%
```

```
  if isa(varargin{j}, 'mfir')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)                  && ...
     isnumeric (obj.fs)                    && ...
     isnumeric (obj.ntaps)                 && ...
     isnumeric (obj.a)                     && ...
     isnumeric (obj.gd)                    && ...
     isnumeric (obj.gain)                  && ...
     isnumeric (obj.histout)               && ...
     ischar    (obj.infile)                && ...
     isa       (obj.plist, 'plist')        && ...
     ischar    (obj.version)               && ...
     isa       (obj.created, 'time')

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for param objects.

SYNOPSIS ^

function result = ne(p1,p2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for param objects.

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

 DESCRIPTION: NE overloads the ~= operator for param objects.

 CALL:        result = ne(p1,p2)
              result = ne(p1,p2, 'property1', 'property2')
              result = ne(p1,p2, 'class/property', 'class/property')
              result = ne(p1,p2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      p1,p2 - input param objects

 OUTPUTS:     If the two param objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.4 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- eq EQ overloads the == operator for param objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(p1,p2, varargin)
% NE overloads the ~= operator for param objects.
%
%%%
```

```
if eq(p1, p2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isvalid

PURPOSE ^

ISVALID tests if the given miir has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given miir has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given miir has all the correct fields of the
              correct type.

 CALL:        isvalid(miir);

              Returns 1 is miir is a valid MIIR-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(miir, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(miir, 'Version')

              The following call returns the category:
              >> ver = isvalid(miir, 'Category')

 VERSION:     $Id: isvalid.m,v 1.2 2008/02/15 17:36:45 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given miir has all the correct fields of the correct type.
%
%%%
```

```
  if isa(varargin{j}, 'miir')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)                  && ...
     isnumeric (obj.fs)                    && ...
     isnumeric (obj.ntaps)                 && ...
     isnumeric (obj.a)                     && ...
     isnumeric (obj.b)                     && ...
     isnumeric (obj.gain)                  && ...
     isnumeric (obj.histin)                && ...
     isnumeric (obj.histout)               && ...
     ischar    (obj.infile)                && ...
     isa       (obj.plist, 'plist')        && ...
     ischar    (obj.version)               && ...
     isa       (obj.created, 'time')

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsasgn

PURPOSE ^

SUBSASGN define index assignment for time properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for time properties.

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

 DESCRIPTION: SUBSASGN define index assignment for time properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:32 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a time object into a string.
- time TIME time object class constructor.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for time properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSASGN define index assignment for time properties.
```

```
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:32 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

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 1.5 2008/02/23 19:44:45 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 1.5 2008/02/23 19:44:45 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
  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

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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isvalid

PURPOSE ^

ISVALID tests if the given param has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given param has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given param has all the correct fields of the
              correct type.

 CALL:        isvalid(param);

              Returns 1 is param is a valid PARAM-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(param, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(param, 'Version')

              The following call returns the category:
              >> ver = isvalid(param, 'Category')

 VERSION:     $Id: isvalid.m,v 1.1 2008/02/14 13:48:33 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given param has all the correct fields of the correct type.
%
%%%
```

```
objs = [];
for j=1:nargin
  if isa(varargin{j}, 'param')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)             && ...
     ischar    (obj.key)              && ...
     ischar    (obj.version)          && ...
     isa       (obj.created, 'time')

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for pole objects.

SYNOPSIS ^

function result = ne(p1, p2 , varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for pole objects.

%%%
```

This function is called by:

SOURCE CODE ^

```
function result = ne(p1, p2 , varargin)
% NE overloads the ~= operator for pole objects.
%
%%%
```

```
if eq(p1, p2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsasgn

PURPOSE ^

SUBSASGN define index assignment for timeformat properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for timeformat properties.

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

 DESCRIPTION: SUBSASGN define index assignment for timeformat properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:32 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a timeformat object into a string.
- timeformat TIMEFORMAT timeformat object class constructor.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for timeformat properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSASGN define index assignment for timeformat properties.
```

```
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:32 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for provenance objects.

SYNOPSIS ^

function result = ne(p1, p2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for provenance objects.

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

 DESCRIPTION: NE overloads the ~= operator for provenance objects.

 CALL:        result = ne(p1,p2)
              result = ne(p1,p2, 'property1', 'property2')
              result = ne(p1,p2, 'class/property', 'class/property')
              result = ne(p1,p2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      p1,p2 - input provenance objects

 OUTPUTS:     If the two provenance objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.4 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a provenance object into a string.
- eq EQ overloads the == operator for provenance objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(p1, p2, varargin)
% NE overloads the ~= operator for provenance objects.
%
%%%
```

```
if eq(p1, p2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsasgn

PURPOSE ^

SUBSASGN define index assignment for timespan properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for timespan properties.

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

 DESCRIPTION: SUBSASGN define index assignment for timespan properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:32 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a timespan object into a string.
- timespan TIMESPAN timespan object class constructor.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for timespan properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSASGN define index assignment for timespan properties.
```

```
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:32 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

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 1.6 2008/02/24 10:17:30 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 1.6 2008/02/24 10:17:30 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);

  for l=1:numel(obj)
    objs = [objs {obj(l)}];
    k = k + 1;
  end
end

% Submit objects
try
  [ids, cid] = ltpda_obj_submit(objs, sinfo);
  if isempty(ids)
    warning('!!! Failed to submit object: %s', objname);
  else
    disp(sprintf('++ Submitted object(s) %s (ref id = %s, collection id = %d)', objname, mat2str(ids), cid));
  end
catch
  warning('!!! Failed to submit object: %s', objname);
  rethrow(lasterror)
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isvalid

PURPOSE ^

ISVALID tests if the given pole has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given pole has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given pole has all the correct fields of the

              correct type.

 CALL:        isvalid(pole);

              Returns 1 is pole is a valid POLE-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(pole, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(pole, 'Version')

              The following call returns the category:
              >> ver = isvalid(pole, 'Category')

 VERSION:     $Id: isvalid.m,v 1.1 2008/02/14 13:48:34 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given pole has all the correct fields of the correct type.
%
%%%
```

```
objs = [];
for j=1:nargin
  if isa(varargin{j}, 'pole')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)                && ...
     isnumeric (obj.f)                   && ...
     isnumeric (obj.q)                   && ...
     isnumeric (obj.ri)                  && ...
     ischar    (obj.version)             && ...
     isa       (obj.created, 'time')     && ...
       (isa (obj.plist, 'plist')    || ...
        isempty(obj.plist))

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for pzmodel objects.

SYNOPSIS ^

function result = ne(p1, p2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for pzmodel objects.

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

 DESCRIPTION: NE overloads the ~= operator for pzmodel objects.

 CALL:        result = ne(p1,p2)
              result = ne(p1,p2,  exc_list)
              result = ne(p1,p2, 'property1', 'property2')
              result = ne(p1,p2, 'class/property', 'class/property')
              result = ne(p1,p2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      p1,p2 - input pzmodel objects

 OUTPUTS:     If the two pzmodel objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.5 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pzmodel object into a string.
- eq EQ overloads the == operator for pzmodel objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(p1, p2, varargin)
% NE overloads the ~= operator for pzmodel objects.
%
%%%
```

```

if eq(p1, p2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsasgn

PURPOSE ^

SUBSASGN define index assignment for for tsdata properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for for tsdata properties.

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

 DESCRIPTION: SUBSASGN define index assignment for tsdata properties.

 EXAMPLES:    All possible assignments are possible.

 VERSION:     $Id: subsasgn.m,v 1.6 2008/02/25 18:06:32 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- tsdata TSDATA time-series object class constructor.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for for tsdata properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSASGN define index assignment for tsdata properties.
```

```
%
% EXAMPLES:    All possible assignments are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.6 2008/02/25 18:06:32 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isvalid

PURPOSE ^

ISVALID tests if the given provenance has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given provenance has all the correct fields of the correct
type.

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

 DESCRIPTION: ISVALID tests if the given provenance has all the correct fields of the
              correct type.

 CALL:        isvalid(provenance);

              Returns 1 is provenance is a valid PROVENANCE-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(provenance, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(provenance, 'Version')

              The following call returns the category:
              >> ver = isvalid(provenance, 'Category')

 VERSION:     $Id: isvalid.m,v 1.1 2008/02/14 13:48:34 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given provenance has all the correct fields of the correct type.
%
%%%
```

```
0047
0048 objs = [];
0049 for j=1:nargin
0050   if isa(varargin{j}, 'provenance')
0051     objs = [objs varargin{j}];
0052   end
0053 end
0054
0055 res = zeros(size(objs));
0056 for j=1:numel(objs)
0057   obj = objs(j);
0058   if ischar    (obj.creator)          && ...
0059      isa       (obj.created, 'time')  && ...
0060      ischar    (obj.ip)               && ...
0061      ischar    (obj.hostname)         && ...
0062      ischar    (obj.os)               && ...
0063      ischar    (obj.matlab_version)   && ...
0064      ischar    (obj.sigproc_version)  && ...
0065      ischar    (obj.ltpda_version)
0066
0067     res(j) = 1;
0068   else
0069     res(j) = 0;
0070   end
0071 end
0072
0073 varargout{1} = res;
0074
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for specwin objects.

SYNOPSIS ^

function result = ne(w1, w2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for specwin objects.

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

 DESCRIPTION: NE overloads the ~= operator for specwin objects.

 CALL:        result = ne(w1,w2)
              result = ne(w1,w2, 'property1', 'property2')
              result = ne(w1,w2, 'class/property', 'class/property')
              result = ne(w1,w2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      w1,w2 - input specwin objects

 OUTPUTS:     If the two specwin objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.4 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a specwin object into a string.
- eq EQ overloads the == operator for specfin objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(w1, w2, varargin)
% NE overloads the ~= operator for specwin objects.
%
%%%
```

```
if eq(w1, w2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsasgn

PURPOSE ^

SUBSASGN define index assignment for xydata properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for xydata properties.

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

 DESCRIPTION: SUBSASGN define index assignment for xydata properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.6 2008/02/25 18:06:32 ingo Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- xydata XYDATA X-Y data object class constructor.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for xydata properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSASGN define index assignment for xydata properties.
```

```
%
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.6 2008/02/25 18:06:32 ingo Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isvalid

PURPOSE ^

ISVALID tests if the given pzmodel has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given pzmodel has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given pzmodel has all the correct fields of the
              correct type.

 CALL:        isvalid(pzmodel);

              Returns 1 is pzmodel is a valid PZMODEL-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(pzmodel, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(pzmodel, 'Version')

              The following call returns the category:
              >> ver = isvalid(pzmodel, 'Category')

 VERSION:     $Id: isvalid.m,v 1.1 2008/02/14 13:48:34 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given pzmodel has all the correct fields of the correct type.
%
%%%
```

```
for j=1:nargin
  if isa(varargin{j}, 'pzmodel')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)             && ...
     isnumeric (obj.gain)                && ...
     isa       (obj.poles, 'pole')    && ...
     isa       (obj.zeros, 'zero')    && ...
     isa       (obj.created, 'time')  && ...
     ischar    (obj.version)          && ...
       (isa (obj.plist, 'plist')    || ...
        isempty(obj.plist))

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for time objects.

SYNOPSIS ^

function result = ne(t1, t2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for time objects.

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

 DESCRIPTION: NE overloads the ~= operator for time objects.

 CALL:        result = ne(t1,t2)
              result = ne(t1,t2,  exc_list)
              result = ne(t1,t2, 'property1', 'property2')
              result = ne(t1,t2, 'class/property', 'class/property')
              result = ne(t1,t2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      t1,t2 - input time objects

 OUTPUTS:     If the two time objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.4 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a time object into a string.
- eq EQ overloads the == operator for time objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(t1, t2, varargin)
% NE overloads the ~= operator for time objects.
%
%%%
```

```
if eq(t1, t2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsasgn

PURPOSE ^

SUBSASGN define index assignment for xyzdata properties.

SYNOPSIS ^

function A = subsasgn(A, S, B)

DESCRIPTION ^

```
 SUBSASGN define index assignment for xyzdata properties.

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

 DESCRIPTION: SUBSASGN define index assignment for xyzdata properties.

 EXAMPLES:    All possible assignment are possible.

 VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:32 ingo Exp $

 HISTORY:     24-12-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- xyzdata XZYDATA X-Y-Z data object class constructor.

This function is called by:

SOURCE CODE ^

```
function A = subsasgn(A, S, B)
% SUBSASGN define index assignment for xyzdata properties.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSASGN define index assignment for xyzdata properties.
%
```

```
% EXAMPLES:    All possible assignment are possible.
%
% VERSION:     $Id: subsasgn.m,v 1.4 2008/02/25 18:06:32 ingo Exp $
%
% HISTORY:     24-12-07 M Hewitson
%                 Creation
%
%%%
```

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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isvalid

PURPOSE ^

ISVALID tests if the given specwin has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given specwin has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given specwin has all the correct fields of the
              correct type.

 CALL:        isvalid(specwin);

              Returns 1 is specwin is a valid SPECWIN-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(specwin, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(specwin, 'Version')

              The following call returns the category:
              >> ver = isvalid(specwin, 'Category')

 VERSION:     $Id: isvalid.m,v 1.2 2008/02/17 18:45:56 hewitson Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given specwin has all the correct fields of the correct type.
%
%%%
```

```
for j=1:nargin
  if isa(varargin{j}, 'specwin')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)                  && ...
     isnumeric (obj.alpha)                 && ...
     isnumeric (obj.psll)                  && ...
     isnumeric (obj.rov)                   && ...
     isnumeric (obj.nenbw)                 && ...
     isnumeric (obj.w3db)                  && ...
     isnumeric (obj.flatness)              && ...
     isnumeric (obj.ws)                    && ...
     isnumeric (obj.ws2)                   && ...
     isnumeric (obj.win)                   && ...
       (isa (obj.plist, 'plist')    || ...
        isempty(obj.plist))                && ...
     ischar    (obj.version)               && ...
     ischar    (obj.created)
%      isa       (obj.created, 'time')

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for timeformat objects.

SYNOPSIS ^

function result = ne(varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for timeformat objects.

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

 DESCRIPTION: NE overloads the ~= operator for timeformat objects.

 CALL:        result = ne(tf1,tf2)
              result = ne(tf1,tf2,  exc_list)
              result = ne(tf1,tf2, 'property1', 'property2')
              result = ne(tf1,tf2, 'class/property', 'class/property')
              result = ne(tf1,tf2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      tf1,tf2 - input timeformat objects

 OUTPUTS:     If the two timeformat objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.6 2008/02/18 19:54:21 hewitson Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a timeformat object into a string.
- eq EQ overloads the == operator for timeformat objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(varargin)
% NE overloads the ~= operator for timeformat objects.
%
%%%
```

```

tf1 = varargin{1};
tf2 = varargin{2};
varargin = varargin(3:end);

if eq(tf1, tf2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

cdata

PURPOSE ^

CDATA constant data object class constructor.

SYNOPSIS ^

function c = cdata(varargin)

DESCRIPTION ^

```
 CDATA constant data object class constructor.

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

 DESCRIPTION: CDATA constant data object class constructor.
              Create a constant data object.

     Properties:
       name    - name of data object
       y       - data values
       x       - cell array: could be one descriptive tag per value
       xunits  - this is used to label the x-axis on plots.
       yunits  - this is used to label the y-axis on plots.
       version - version of the constructor code
       created - creation time of this fsdata object.

     Possible constructors:
       c = cdata()       - creates a blank data object
       c = cdata(pl)     - creates a data object with the given
                           parameter list which contains 'N' and 'Val'
       c = cdata(vals)   - creates a data object with the given
                           data values.
       c = cdata(N,v)    - creates a data object with N samples all of
                           value, v.

 HISTORY: 30-01-2007 Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- set SET set a cdata property.

This function is called by:

- subsasgn SUBSASGN define index assignment for fsdata properties.

SUBFUNCTIONS ^

- function c = init()
- function c = checkCdata(c)

SOURCE CODE ^

```
function c = cdata(varargin)
% CDATA constant data object class constructor.
%
%%%
```

```
    c = plist();
    return
  elseif strcmp(varargin{2}, 'Version')
    c = VERSION;
    return
  elseif strcmp(varargin{2}, 'Category')
    c = CATEGORY;
    return
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%   define cdata properties   %%%%%%%%%%%%%%%%%%%%%%%%

  function c = init()
    c.name    = 'None';
    c.y       = [];
    c.x       = {};
    c.xunits  = 'N/A';
    c.yunits  = 'N/A';
    c.version = VERSION;
    c.created   = time;
    c = class(c, 'cdata');
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%   Create cdata object   %%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%  val = cdata()   %%%%%%%%%%
% create default cdata object
if nargin == 0
  c = init();

elseif nargin == 1

  %%%%%%%%%%%% From File %%%%%%%%%%%%%%%%
  if ischar(varargin{1})

    filename = varargin{1};
    [path, name, ext, vers] = fileparts(filename);
    switch ext
      case '.mat'
        c = load(filename);
        c = c.a;
      case '.xml'
        root_node = xmlread(filename);
        c = ltpda_xmlread(root_node, 'cdata');
      otherwise
        error('### Unknown file type.');
    end
  %%%%%%%%%%  val = cdata(cdata)   %%%%%%%%%%
  elseif isa(varargin{1}, 'cdata')
    c = varargin{1};

```

```
    %%%%%%%%%%  val = cdata(struct)   %%%%%%%%%%
  elseif isstruct(varargin{1})

    c = init();

    fields = fieldnames(varargin{1});
    for ii = 1:length(fields)
      field = fields{ii};

      %%% created -> time-object
      if strcmp(field, 'created')
        created       = varargin{1}.created;
        if isstruct(created)
          created = time(created);
        end
        c.created     = created;
      %%% All other
      else
        try
          c.(field) = varargin{1}.(field);
        catch
          error('### The field ''%s'' in the struct is not a cdata property.', field)
        end
      end
    end

    %%%%%%%%%%  val = cdata(plist)   %%%%%%%%%%
  elseif isa(varargin{1}, 'plist')

    c = init();

    % get possible parameters
    fcn  = find(varargin{1}, 'fcn');

    y    = find(varargin{1}, 'vals');
    if isempty(y)
      y = find(varargin{1}, 'y');
    end

    % check which to use
    if ~isempty(fcn)
      % do function constructor
      y = eval(fcn);
    elseif ~isempty(y)
      % do vals constructor
      %- nothing to do here.
    else
      error('### unknown constructor method for cdata.');
    end

    c.name = fcn;
```

```
    c.y    = y;

    %%%%%%%%%%  val = cdata(vals)   %%%%%%%%%%
    %%%%%%%%%%  val = cdata(y)   %%%%%%%%%%
  else

    c      = init();
    c.name = 'Data';
    c.y    = varargin{1};

  end


  %%%%%%%%%%  val = cdata(N,val)   %%%%%%%%%%
elseif nargin == 2

  %%%%%%%%%%% From DATABASE
  if isa(varargin{1}, 'database')
    c = retrieve(varargin{1}, varargin{2:end});
  else
    c = init();
    c.name = 'Constant';
    c.y    = ones(varargin{1},1)*varargin{2};
  end
else
  error('### Unknown number of constructor arguments.');
end

% check data
% c = checkCdata(c);
end


%-----------------------------------------------------------------------
% check vector has right dimensions
function c = checkCdata(c)

x = c.y;
if size(x, 1) < size(x, 2)
  x = x.';
end

c = set(c, 'y', x);
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isvalid

PURPOSE ^

ISVALID tests if the given time has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given time has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given time has all the correct fields of the
              correct type.

 CALL:        isvalid(time);

              Returns 1 is time is a valid TIME-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(time, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(time, 'Version')

              The following call returns the category:
              >> ver = isvalid(time, 'Category')

 VERSION:     $Id: isvalid.m,v 1.1 2008/02/14 13:48:34 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given time has all the correct fields of the correct type.
%
%%%
```

```
  if isa(varargin{j}, 'time')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)                    && ...
     isnumeric (obj.utc_epoch_milli)         && ...
     isjava    (obj.timezone)                && ...
     isa       (obj.timeformat, 'timeformat') && ...
     ischar    (obj.time_str)                && ...
     ischar    (obj.created)                 && ...
     ischar    (obj.version)                 && ...
       (isa (obj.plist, 'plist')    || ...
        isempty(obj.plist))

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for timespan objects.

SYNOPSIS ^

function result = ne(ts1, ts2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for timespan objects.

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

 DESCRIPTION: NE overloads the ~= operator for timespan objects.

 CALL:        result = ne(ts1,ts2)
              result = ne(ts1,ts2,  exc_list)
              result = ne(ts1,ts2, 'property1', 'property2')
              result = ne(ts1,ts2, 'class/property', 'class/property')
              result = ne(ts1,ts2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      ts1,ts2 - input timespan objects

 OUTPUTS:     If the two timespan objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.4 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a timespan object into a string.
- eq EQ overloads the == operator for timespan objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(ts1, ts2, varargin)
% NE overloads the ~= operator for timespan objects.
%
%%%
```

```

if eq(ts1, ts2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isvalid

PURPOSE ^

ISVALID tests if the given timeformat has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given timeformat has all the correct fields of the correct
type.

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

 DESCRIPTION: ISVALID tests if the given timeformat has all the correct fields of the
              correct type.

 CALL:        isvalid(timeformat);

              Returns 1 is timeformat is a valid TIMEFORMAT-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(timeformat, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(timeformat, 'Version')

              The following call returns the category:
              >> ver = isvalid(timeformat, 'Category')

 VERSION:     $Id: isvalid.m,v 1.1 2008/02/14 13:48:34 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given timeformat has all the correct fields of the correct type.
%
%%%
```

```

objs = [];
for j=1:nargin
  if isa(varargin{j}, 'timeformat')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.format_str)         && ...
     isnumeric (obj.format_nr)          && ...
     ischar    (obj.default_matlab_str) && ...
     isnumeric (obj.default_matlab_nr)  && ...
     ischar    (obj.default_java_str)

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for tsdata objects.

SYNOPSIS ^

function result = ne(c1, c2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for tsdata objects.

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

 DESCRIPTION: NE overloads the ~= operator for tsdata objects.

 CALL:        result = ne(c1,c2)
              result = ne(c1,c2, 'property1', 'property2')
              result = ne(c1,c2, 'class/property', 'class/property')
              result = ne(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      c1,c2 - input tsdata objects

 OUTPUTS:     If the two tsdata objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.4 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- eq EQ overloads the == operator for tsdata objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(c1, c2, varargin)
% NE overloads the ~= operator for tsdata objects.
%
%%%
```

```
if eq(c1,c2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for cdata objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for cdata objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for cdata objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.9 2008/02/22 22:26:40 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a cdata object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for cdata objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for cdata objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.9 2008/02/22 22:26:40 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

char

PURPOSE ^

CHAR convert a cdata object into a string.

SYNOPSIS ^

function pstr = char(p, varargin)

DESCRIPTION ^

```
 CHAR convert a cdata object into a string.

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

 DESCRIPTION: CHAR convert a cdata object into a string.

 VERSION: $Id: char.m,v 1.4 2008/02/12 19:54:05 hewitson Exp $

 HISTORY: 04-02-2007 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a cdata object into a string.
- get GET get csdata properties.

This function is called by:

- char CHAR convert a cdata object into a string.
- display DISPLAY implement terminal display for cdata object.
- get_xy_values GET_XY_VALUES returns the values in vals.
- ne NE overloads the ~= operator for cdata objects.
- save SAVE a cdata object to file.
- set_xy_axis SET_XY_AXIS set the x-axis and y-axis values.
- single_operation SINGLE_OPERATION implements specified operator overload for cdata objects.
- subsasgn SUBSASGN define index assignment for fsdata properties.
- subsref SUBSREF Define field name indexing for cdata objects.

SOURCE CODE ^

```
function pstr = char(p, varargin)
% CHAR convert a cdata object into a string.
%
%%%
```

isvalid

PURPOSE ^

ISVALID tests if the given timespan has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given timespan has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given timespan has all the correct fields of the
              correct type.

 CALL:        isvalid(timespan);

              Returns 1 is timespan is a valid TIMESPAN-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(timespan, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(timespan, 'Version')

              The following call returns the category:
              >> ver = isvalid(timespan, 'Category')

 VERSION:     $Id: isvalid.m,v 1.1 2008/02/14 13:48:34 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given timespan has all the correct fields of the correct type.
%
%%%
```

```
for j=1:nargin
  if isa(varargin{j}, 'timespan')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)                    && ...
     isa       (obj.start, 'time')           && ...
     isa       (obj.end, 'time')             && ...
     isa       (obj.timeformat, 'timeformat') && ...
     ischar    (obj.interval)                && ...
     isjava    (obj.timezone)                && ...
     ischar    (obj.version)                 && ...
       (isa (obj.plist, 'plist')    || ...
        isempty(obj.plist))

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for xydata objects.

SYNOPSIS ^

function result = ne(c1, c2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for xydata objects.

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

 DESCRIPTION: NE overloads the ~= operator for xydata objects.

 CALL:        result = ne(c1,c2)
              result = ne(c1,c2, 'property1', 'property2')
              result = ne(c1,c2, 'class/property', 'class/property')
              result = ne(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      c1,c2 - input xydata objects

 OUTPUTS:     If the two xydata objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.4 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- eq EQ overloads the == operator for xydata objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(c1, c2, varargin)
% NE overloads the ~= operator for xydata objects.
%
%%%
```

```
if eq(c1,c2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for fsdata objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for fsdata objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for fsdata objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.15 2008/02/22 22:26:40 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for fsdata objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for fsdata objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.15 2008/02/22 22:26:40 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

char

PURPOSE ^

CHAR convert a param object into a string.

SYNOPSIS ^

function pstr = char(p, varargin)

DESCRIPTION ^

```
 CHAR convert a param object into a string.

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

 DESCRIPTION: CHAR convert a param object into a string.

 CALL:        string = char(fsd)

 INPUTS:      fsd    - frequency-series object

 OUTPUTS:     string - a string which represent a frequency-series object

 VERSION:     $Id: char.m,v 1.4 2008/02/12 22:11:21 hewitson Exp $

 HISTORY:     04-02-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- get GET get a fsdata property.

This function is called by:

- char CHAR convert a param object into a string.
- display DISPLAY implement terminal display for fsdata object.
- get_xy_values GET_XY_VALUES returns the x-axis and/or y-axis values.
- ne NE overloads the ~= operator for fsdata objects.

- save SAVE an fsdata object to file.
- set_xy_axis SET_XY_AXIS set the x-axis and y-axis values.
- single_operation SINGLE_OPERATION implements specified operator overload for fsdata objects.
- subsasgn SUBSASGN define index assignment for fsdata properties.
- subsref SUBSREF Define field name indexing for fsdata objects.

SOURCE CODE ^

```
function pstr = char(p, varargin)
% CHAR convert a param object into a string.
%
%%%
```

```
0043
0044 % END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isvalid

PURPOSE ^

ISVALID tests if the given tsdata has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given tsdata has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given tsdata has all the correct fields of the
              correct type.

 CALL:        isvalid(tsdata);

              Returns 1 is tsdata is a valid TSDATA-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(tsdata, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(tsdata, 'Version')

              The following call returns the category:
              >> ver = isvalid(tsdata, 'Category')

 VERSION:     $Id: isvalid.m,v 1.1 2008/02/14 13:48:34 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given tsdata has all the correct fields of the correct type.
%
%%%
```

```
objs = [];
for j=1:nargin
  if isa(varargin{j}, 'tsdata')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)             && ...
     isnumeric (obj.y)                && ...
     isnumeric (obj.x)                && ...
     isnumeric (obj.fs)               && ...
     ischar    (obj.xunits)           && ...
     ischar    (obj.yunits)           && ...
     isnumeric (obj.nsecs)            && ...
     isa       (obj.t0, 'time')       && ...
     ischar    (obj.version)          && ...
     isa       (obj.created, 'time')

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for xyzdata objects.

SYNOPSIS ^

function result = ne(c1, c2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for xyzdata objects.

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

 DESCRIPTION: NE overloads the ~= operator for xyzdata objects.

 CALL:        result = ne(c1,c2)
              result = ne(c1,c2, 'property1', 'property2')
              result = ne(c1,c2, 'class/property', 'class/property')
              result = ne(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      c1,c2 - input xyzdata objects

 OUTPUTS:     If the two xyzdata objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.3 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     24-12-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- eq EQ overloads the == operator for xyzdata objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(c1, c2, varargin)
% NE overloads the ~= operator for xyzdata objects.
%
%%%
```

```
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for history objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for history objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for history objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.5 2008/02/22 22:26:41 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for history objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for history objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.5 2008/02/22 22:26:41 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

char

PURPOSE ^

CHAR convert a param object into a string.

SYNOPSIS ^

function pstr = char(p, varargin)

DESCRIPTION ^

```
 CHAR convert a param object into a string.

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

 DESCRIPTION: CHAR convert a param object into a string.

 CALL:        string = char(hist)

 VERSION:     $Id: char.m,v 1.4 2008/02/12 22:30:27 hewitson Exp $

 HISTORY:     04-02-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- get GET get history object properties.

This function is called by:

- char CHAR convert a param object into a string.
- 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.
- ne NE overloads the ~= operator for history objects.

- plot PLOT plots a history object as a tree diagram.
- save SAVE a history object to file.
- string STRING writes a command string that can be used to recreate the input history object.
- subsasgn SUBSASGN define index assignment for history properties.
- subsref SUBSREF Define field name indexing for history objects.

SOURCE CODE ^

```
function pstr = char(p, varargin)
% CHAR convert a param object into a string.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: CHAR convert a param object into a string.
%
% CALL:        string = char(hist)
%
% VERSION:     $Id: char.m,v 1.4 2008/02/12 22:30:27 hewitson Exp $
%
% HISTORY:     04-02-2007 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: char.m,v 1.4 2008/02/12 22:30:27 hewitson Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(p, 'history') && ischar(varargin{1})
    in = char(varargin{1});
    if strcmp(in, 'Params')
      pstr = plist();
      return
    elseif strcmp(in, 'Version')
      pstr = VERSION;
      return
    elseif strcmp(in, 'Category')
      pstr = CATEGORY;
      return
    end
  end
end

pstr = get(p,'name');

% END
```

isvalid

PURPOSE ^

ISVALID tests if the given xydata has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given xydata has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given xydata has all the correct fields of the
              correct type.

 CALL:        isvalid(xydata);

              Returns 1 is xydata is a valid XYDATA-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(xydata, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(xydata, 'Version')

              The following call returns the category:
              >> ver = isvalid(xydata, 'Category')

 VERSION:     $Id: isvalid.m,v 1.1 2008/02/14 13:48:34 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given xydata has all the correct fields of the correct type.
%
%%%
```

```
objs = [];
for j=1:nargin
  if isa(varargin{j}, 'xydata')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)             && ...
     isnumeric (obj.y)                && ...
     isnumeric (obj.x)                && ...
     ischar    (obj.xunits)           && ...
     ischar    (obj.yunits)           && ...
     ischar    (obj.version)          && ...
     isa       (obj.created, 'time')

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ne

PURPOSE ^

NE overloads the ~= operator for zero objects.

SYNOPSIS ^

function result = ne(p1, p2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for zero objects.

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

 DESCRIPTION: NE overloads the ~= operator for zero objects.

 CALL:        result = ne(z1,z1)
              result = ne(p1,p2, 'property1', 'property2')
              result = ne(p1,p2, 'class/property', 'class/property')
              result = ne(p1,p2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 INPUTS:      z1,z2 - input zero objects

 OUTPUTS:     If the two zero objects are considered equal, result == 0,
              otherwise, result == 1.

 VERSION:     $Id: ne.m,v 1.4 2008/02/18 15:12:58 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a zero object into a string.
- eq EQ overloads the == operator for zero objects.

This function is called by:

SOURCE CODE ^

```
function result = ne(p1, p2, varargin)
% NE overloads the ~= operator for zero objects.
%
%%%
```

```
if eq(p1, p2, varargin{:})
  result = 0;
else
  result = 1;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for mfir objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for mfir objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for mfir objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a mfir object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for mfir objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for mfir objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

char

PURPOSE ^

CHAR convert a mfir object into a string.

SYNOPSIS ^

function pstr = char(w, varargin)

DESCRIPTION ^

```
 CHAR convert a mfir object into a string.

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

 DESCRIPTION: CHAR convert a mfir object into a string.

 CALL:        string = char(mfir)

 VERSION:     $Id: char.m,v 1.4 2008/02/12 22:39:49 hewitson Exp $

 HISTORY:     04-02-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a mfir object into a string.

This function is called by:

- char CHAR convert a mfir object into a string.
- display DISPLAY overloads display functionality for mfir objects.
- ne NE overloads the ~= operator for mfir objects.
- redesign REDESIGN redesign the input filter to work for the given sample rate.
- resp RESP Make a frequency response of the filter.
- save SAVE an mfir object to file.
- string STRING writes a command string that can be used to recreate the input filter object.
- subsasgn SUBSASGN define index assignment for mfir properties.
- subsref SUBSREF Define field name indexing for mfir objects.

SOURCE CODE ^

```
function pstr = char(w, varargin)
% CHAR convert a mfir object into a string.
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isvalid

PURPOSE ^

ISVALID tests if the given xyzdata has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given xyzdata has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given xyzdata has all the correct fields of the
              correct type.

 CALL:        isvalid(xyzdata);

              Returns 1 is xyzdata is a valid XYZDATA-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(xyzdata, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(xyzdata, 'Version')

              The following call returns the category:
              >> ver = isvalid(xyzdata, 'Category')

 VERSION:     $Id: isvalid.m,v 1.1 2008/02/14 13:48:34 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given xyzdata has all the correct fields of the correct type.
%
%%%
```

```
for j=1:nargin
  if isa(varargin{j}, 'xyzdata')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)            && ...
     isnumeric (obj.y)               && ...
     isnumeric (obj.x)               && ...
     isnumeric (obj.z)               && ...
     ischar    (obj.xunits)          && ...
     ischar    (obj.yunits)          && ...
     ischar    (obj.zunits)          && ...
     ischar    (obj.version)         && ...
     isa       (obj.created, 'time')

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ngconv

PURPOSE ^

NGINIT is called by the function LTPDA_NOISEGEN

SYNOPSIS ^

function [a,b] = ngconv(pzm)

DESCRIPTION ^

```
NGINIT 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 1.2 2008/01/23 17:23:56 anneke Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpda_noisegen LTPDA_NOISEGEN uses pzm2ab and franklin to generate a time-series

SOURCE CODE ^

```
function [a,b] = ngconv(pzm)
% NGINIT 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 1.2 2008/01/23 17:23:56 anneke Exp $



zs = get(pzm, 'zeros');
ps = get(pzm, 'poles');

%%
f_zer = [];
q_zer = [];
for j=1:length(zs)
    z = zs(j);
    f_zer(j,1) = [get(z,'f')];
    q_zer(j,1) = [get(z,'q')];
    %if isnan(q_zer(j))
    %    q_zer(j,1) = 0;
    %end
    %zv(j,1:2) = [f q];
end

f_pol = [];
q_pol = [];
for j=1:length(ps)
    p = ps(j);
    f_pol(j,1) = [get(p,'f')];
    q_pol(j,1) = [get(p,'q')];
    %if isnan(q_pol(j))
    %    q_pol(j,1) = 0;
    %end
end
%% calculate factors from f and q
pol = fq2fac(f_pol,q_pol);
zer = fq2fac(f_zer,q_zer);
%%
[b,a] = conv_noisegen(pol,zer);

```

```
%END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for miir objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for miir objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for miir objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a miir object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for miir objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for miir objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

char

PURPOSE ^

CHAR convert a miir object into a string.

SYNOPSIS ^

function pstr = char(w, varargin)

DESCRIPTION ^

```
 CHAR convert a miir object into a string.

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

 DESCRIPTION: CHAR convert a miir object into a string.

 CALL:        string = char(miir)

 VERSION:     $Id: char.m,v 1.5 2008/02/12 22:46:55 hewitson Exp $

 HISTORY:     04-02-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a miir object into a string.

This function is called by:

- char CHAR convert a miir object into a string.
- display DISPLAY overloads display functionality for miir objects.
- ne NE overloads the ~= operator for miir objects.
- redesign REDESIGN redesign the input filter to work for the given sample rate.
- resp RESP Make a frequency response of the filter.
- save SAVE an miir object to file.
- string STRING writes a command string that can be used to recreate the input filter object.
- subsasgn SUBSASGN define index assignment for miir properties.
- subsref SUBSREF Define field name indexing for miir objects.

SOURCE CODE ^

```
function pstr = char(w, varargin)
% CHAR convert a miir object into a string.
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

isvalid

PURPOSE ^

ISVALID tests if the given zero has all the correct fields of the correct type.

SYNOPSIS ^

function varargout = isvalid(varargin)

DESCRIPTION ^

```
 ISVALID tests if the given zero has all the correct fields of the correct type.

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

 DESCRIPTION: ISVALID tests if the given zero has all the correct fields of the
              correct type.

 CALL:        isvalid(zero);

              Returns 1 is zero is a valid ZERO-object; 0 otherwise.

              The following call returns a parameter list object that
              contains the default parameter values:
              >> pl = isvalid(zero, 'Params')

              The following call returns the cvs version number:
              >> ver = isvalid(zero, 'Version')

              The following call returns the category:
              >> ver = isvalid(zero, 'Category')

 VERSION:     $Id: isvalid.m,v 1.1 2008/02/14 13:48:34 ingo Exp $

 HISTORY:     14-02-2008 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = isvalid(varargin)
% ISVALID tests if the given zero has all the correct fields of the correct type.
%
%%%
```

```
  if isa(varargin{j}, 'zero')
    objs = [objs varargin{j}];
  end
end

res = zeros(size(objs));
for j=1:numel(objs)
  obj = objs(j);
  if ischar    (obj.name)                && ...
     isnumeric (obj.f)                   && ...
     isnumeric (obj.q)                   && ...
     isnumeric (obj.ri)                  && ...
     ischar    (obj.version)             && ...
     isa       (obj.created, 'time')     && ...
       (isa (obj.plist, 'plist')    || ...
        isempty(obj.plist))

    res(j) = 1;
  else
    res(j) = 0;
  end
end

varargout{1} = res;

```

Generated on Tue 26-Feb-2008 10:52:52 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 2008/01/23 17:23:56 anneke Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpda_noisegen LTPDA_NOISEGEN uses pzm2ab and franklin to generate a time-series

SOURCE CODE ^

```
function y = nginit(Tinit)
% 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 2008/01/23 17:23:56 anneke Exp $

n = length(Tinit);
%% writing the generator
r = randn(n,1);
y = Tinit * r;
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for parameter objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for parameter objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for parameter objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.9 2008/02/22 22:26:40 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for parameter objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for parameter objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.9 2008/02/22 22:26:40 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

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 1.4 2008/01/23 17:23:56 anneke Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpda_noisegen LTPDA_NOISEGEN uses pzm2ab and franklin to generate a time-series

SOURCE CODE ^

```
function [x y] = ngprop(Tprop, E, num, y, ns)
% 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 1.4 2008/01/23 17:23:56 anneke Exp $

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


```

Generated on Tue 26-Feb-2008 10:52:52 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 2007/05/19 09:04:59 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- get_window GET_WINDOW returns the required window function as a structure.

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 2007/05/19 09:04:59 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 Tue 26-Feb-2008 10:52:52 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 2008/01/23 17:23:56 anneke Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpda_noisegen LTPDA_NOISEGEN uses pzm2ab and franklin to generate a time-series

SOURCE CODE ^

```
function [Tinit,Tprop,E] = ngsetup(den,fs)
% 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 2008/01/23 17:23:56 anneke Exp $


den=den';
dt = 1/fs;
length(den);

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
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for pole objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for pole objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for pole objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pole object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for pole objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for pole objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

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 2007/05/19 09:04:59 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- get_window GET_WINDOW returns the required window function as a structure.

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 2007/05/19 09:04:59 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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for provenance objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for provenance objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for provenance objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.5 2008/02/22 22:26:41 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a provenance object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for provenance objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for provenance objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.5 2008/02/22 22:26:41 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

char

PURPOSE ^

CHAR convert a provenance object into a string.

SYNOPSIS ^

function pstr = char(p, varargin)

DESCRIPTION ^

```
 CHAR convert a provenance object into a string.

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

 DESCRIPTION: CHAR convert a provenance object into a string.

 CALL:        string = char(prov)

 VERSION:     $Id: char.m,v 1.6 2008/02/13 17:11:03 mauro Exp $

 HISTORY:     04-02-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a provenance object into a string.
- get GET get provenance properties.

This function is called by:

- char CHAR convert a provenance object into a string.
- display DISPLAY overload terminal display for provenance objects.
- ne NE overloads the ~= operator for provenance objects.
- provenance PROVENANCE constructors for provenance class.
- save SAVE a provenance object to file.
- string STRING writes a command string that can be used to recreate the input provenance object.
- subsasgn SUBSASGN define index assignment for provenance properties.
- subsref SUBSREF Define field name indexing for provenance objects.

SOURCE CODE ^

```
function pstr = char(p, varargin)
% CHAR convert a provenance object into a string.
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 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 2007/05/19 09:04:59 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- get_window GET_WINDOW returns the required window function as a structure.

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 2007/05/19 09:04:59 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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for pole/zero model objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for pole/zero model objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for pole/zero model objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pzmodel object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for pole/zero model objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for pole/zero model objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

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 2007/05/19 09:04:59 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- get_window GET_WINDOW returns the required window function as a structure.

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 2007/05/19 09:04:59 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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for specwin objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for specwin objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for specwin objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.5 2008/02/22 22:26:41 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a specwin object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for specwin objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for specwin objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.5 2008/02/22 22:26:41 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

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 2007/05/19 09:04:59 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- get_window GET_WINDOW returns the required window function as a structure.

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 2007/05/19 09:04:59 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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

parseFilterParams

PURPOSE ^

PARSEFILTERPARAMS parses the input plist and returns a full plist for

SYNOPSIS ^

function plo = parseFilterParams(pl)

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.

M Hewitson 11-02-09



$Id: parseFilterParams.m,v 1.3 2008/02/11 12:31:11 mauro Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function plo = parseFilterParams(pl)

% 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.
%
% M Hewitson 11-02-09
%
%
% $Id: parseFilterParams.m,v 1.3 2008/02/11 12:31:11 mauro Exp $
%

%   'type'  - one of 'highpass', 'lowpass', 'bandpass', 'bandreject'.
%             [default: 'lowpass']
```

```
%   'gain'  - gain of filter [default: 1.0]
%   'fs'    - sample frequency to design for [default: 1Hz]
%   'order' - order of filter [default: 1+1]
%   'fc'    - corner frequencies. This is a two element vector for bandpass
%             and bandreject filters. [default: 0.25 or [0.1 0.2] ]
%   'Win'   - a window object to use in the design. [default: Hamming]
%
%

plo = plist();

% type
type = find(pl, 'type');
if isempty(type)
  type = 'lowpass';
  disp('- using default type ''lowpass''');
end
plo = append(plo, param('type', type));

% gain
gain = find(pl, 'gain');
if isempty(gain)
  gain = 1.0;
  disp(['- using default gain ' num2str(gain)]);
end
plo = append(plo, param('gain', gain));

% order
order = find(pl, 'order');
if isempty(order)
  order = 1;
  disp(['- using default order ' num2str(order)]);
end
if mod(order,2) == 1
  warning('!!! reseting filter order to even number (+1)')
  order = order + 1;
end
plo = append(plo, param('order', order));

% fc
fc = find(pl, 'fc');
if isempty(fc)
  if strcmp(type, 'bandreject') | strcmp(type, 'bandpass')
    fc = [0.1 0.2];
  else
    fc = 0.25;
  end
  disp(['- using default fc ' num2str(fc)]);
end
plo = append(plo, param('fc', fc));

% fs
```

```
fs = find(pl, 'fs');
if isempty(fs)
  fs = 10*max(fc);
  warning([sprintf('!!! no sample rate specified. Designing for fs=%2.2fHz.', fs)...
            sprintf('\nThe filter will be redesigned later when used.')]);
end
plo = append(plo, param('fs', fs));

% win
win = find(pl, 'Win');
if isempty(win)
  % then we use the default window
  win = specwin('Hamming', order+1);
end
if length(win.win) ~= order + 1
  warning('!!! setting window length to filter order !!!');
  switch win.name
    case {'Kaiser', 'Flattop'}
      win = specwin(win.name, order + 1, win.psll);
    otherwise
      win = specwin(win.name, order + 1);
  end
end
plo = append(plo, param('Win', win));
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for time objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for time objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for time objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.9 2008/02/22 22:26:40 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a time object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for time objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for time objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.9 2008/02/22 22:26:40 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

char

PURPOSE ^

CHAR convert a time object into a string.

SYNOPSIS ^

function pstr = char(tt, varargin)

DESCRIPTION ^

```
 CHAR convert a time object into a string.

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

 DESCRIPTION: CHAR convert a time object into a string.

 CALL:        string = char(time)

 VERSION:     $Id: char.m,v 1.5 2008/02/13 16:31:26 mauro Exp $

 HISTORY:     23-07-2007 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a time object into a string.

This function is called by:

- char CHAR convert a time object into a string.
- display DISPLAY overloads display functionality for time objects.
- format FORMAT Returns the time in the handover format.
- get GET get time properties.
- minus MINUS overloads - operator for time objects.
- ne NE overloads the ~= operator for time objects.
- plus PLUS overloads - operator for time objects.
- save SAVE a time object to file.
- set SET set time properties.

- string STRING writes a command string that can be used to recreate the input time object.
- subsasgn SUBSASGN define index assignment for time properties.
- subsref SUBSREF Define field name indexing for time objects.
- time TIME time object class constructor.

SOURCE CODE ^

```
function pstr = char(tt, varargin)
% CHAR convert a time object into a string.
%
%%%
```

parseFilterParams

PURPOSE ^

PARSEFILTERPARAMS parses the input plist and returns a full plist for

SYNOPSIS ^

function plo = parseFilterParams(pl)

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.

M Hewitson 11-02-09



$Id: parseFilterParams.m,v 1.6 2008/02/11 11:38:40 mauro Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function plo = parseFilterParams(pl)

% 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.
%
% M Hewitson 11-02-09
%
%
% $Id: parseFilterParams.m,v 1.6 2008/02/11 11:38:40 mauro Exp $
%

%   'type'  - one of 'highpass', 'lowpass', 'bandpass', 'bandreject'.
%             [default: 'lowpass']
```

```
%   'gain'  - gain of filter [default: 1.0]
%   'fs'    - sample frequency to design for [default: 1Hz]
%   'order' - order of filter [default: 1]
%   'fc'    - corner frequencies. This is a two element vector for bandpass
%             and bandreject filters. [default: 0.25 or [0.1 0.2] ]

plo = plist();

% type
type = find(pl, 'type');
if isempty(type)
  type = 'lowpass';
  disp('- using default type ''lowpass''');
end
plo = append(plo, param('type', type));

% gain
gain = find(pl, 'gain');
if isempty(gain)
  gain = 1.0;
  disp(['- using default gain ' num2str(gain)]);
end
plo = append(plo, param('gain', gain));

% order
order = find(pl, 'order');
if isempty(order)
  order = 1.0;
  disp(['- using default order ' num2str(order)]);
end
plo = append(plo, param('order', order));

% fc
fc = find(pl, 'fc');
if isempty(fc)
  if strcmp(type, 'bandreject') | strcmp(type, 'bandpass')
    fc = [0.1 0.2];
  else
    fc = 0.25;
  end
  disp(['- using default fc ' num2str(fc)]);
end
plo = append(plo, param('fc', fc));

% fs
fs = find(pl, 'fs');
if isempty(fs)
  fs = 10*max(fc);
  warning([sprintf('!!! no sample rate specified. Designing for fs=%2.2fHz.', fs)...
           sprintf('\nThe filter will be redesigned later when used.')]);
end
```

```
plo = append(plo, param('fs', fs));

% ripple
ripple = find(pl, 'ripple');
if isempty(ripple)
  ripple = 0.5;
  disp(['- using default ripple ' num2str(ripple)]);
end
plo = append(plo, param('ripple', ripple));
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for timeformat objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for timeformat objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for timeformat objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a timeformat object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for timeformat objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for timeformat objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

char

PURPOSE ^

CHAR convert a timeformat object into a string.

SYNOPSIS ^

function pstr = char(tf, varargin)

DESCRIPTION ^

```
 CHAR convert a timeformat object into a string.

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

 DESCRIPTION: CHAR convert a timeformat object into a string.

 CALL:        string = char(timeformat)

 VERSION:     $Id: char.m,v 1.3 2008/02/13 16:54:14 mauro Exp $

 HISTORY:     23-07-2007 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a timeformat object into a string.

This function is called by:

- char CHAR convert a timeformat object into a string.
- display DISPLAY overloads display functionality for timeformat objects.
- ne NE overloads the ~= operator for timeformat objects.
- save SAVE a timeformat object to file.
- string STRING writes a command string that can be used to recreate the input timeformat object.
- subsasgn SUBSASGN define index assignment for timeformat properties.
- subsref SUBSREF Define field name indexing for timeformat objects.
- timeformat TIMEFORMAT timeformat object class constructor.

SOURCE CODE ^

```
function pstr = char(tf, varargin)
% CHAR convert a timeformat object into a string.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: CHAR convert a timeformat object into a string.
%
% CALL:        string = char(timeformat)
%
% VERSION:     $Id: char.m,v 1.3 2008/02/13 16:54:14 mauro Exp $
%
% HISTORY:     23-07-2007 Diepholz
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION = '$Id: char.m,v 1.3 2008/02/13 16:54:14 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters or for the cvs-version number
if nargin == 2
  if isa(tf, 'timeformat') && ischar(varargin{1})
    in = char(varargin{1});
    if strcmp(in, 'Params')
      pstr = plist();
      return
    elseif strcmp(in, 'Version')
      pstr = VERSION;
      return
    elseif strcmp(in, 'Category')
      pstr = CATEGORY;
      return
    end
  end
end

if ~isempty(tf.format_str)
  pstr = tf.format_str;
else
  pstr = num2str(tf.format_nr);
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

pfq2ri

PURPOSE ^

Convert frequency/Q pole/zero representation into real

SYNOPSIS ^

function ri= pfq2ri(f0, Q)

DESCRIPTION ^

```
Convert frequency/Q pole/zero representation into real
and imaginary conjugate pairs. Returns [ri conj(ri)]

ri= pfq2ri(f0, Q)

M Hewitson 02-04-07


$Id: pfq2ri.m,v 1.10 2008/02/13 10:13:54 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ri= pfq2ri(f0, Q)

% Convert frequency/Q pole/zero representation into real
% and imaginary conjugate pairs. Returns [ri conj(ri)]
%
```

```
% ri= pfq2ri(f0, Q)
%
% M Hewitson 02-04-07
%
%
% $Id: pfq2ri.m,v 1.10 2008/02/13 10:13:54 ingo Exp $
%

if(nargin==0)
  disp('usage: ri = fq2ri2(f0, Q)')
  return
elseif(nargin==1 || nargin==2 && isnan(Q))
    ri = -(1/(2*pi*f0));
elseif(nargin==2 && Q>=0)
    if Q < 0.5
      disp('Q < 0.5! Splitting to two real poles;')
      ri = [-(1/(2*pi*f0)) -(1/(2*pi*f0))];
    else
      w0  = 2*pi*f0;
      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


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for timespan objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for timespan objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for timespan objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a timespan object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for timespan objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for timespan objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

char

PURPOSE ^

CHAR convert a timespan object into a string.

SYNOPSIS ^

function pstr = char(ts ,varargin)

DESCRIPTION ^

```
 CHAR convert a timespan object into a string.

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

 DESCRIPTION: CHAR convert a timespan object into a string.

 CALL:        string = char(timespan)

 VERSION:     $Id: char.m,v 1.3 2008/02/13 16:42:51 mauro Exp $

 HISTORY:     23-07-2007 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a timespan object into a string.

This function is called by:

- char CHAR convert a timespan object into a string.
- display DISPLAY overloads display functionality for timespan objects.
- ne NE overloads the ~= operator for timespan objects.
- save SAVE a timespan object to file.
- set SET set timespan properties.
- string STRING writes a command string that can be used to recreate the input timespan object.
- subsasgn SUBSASGN define index assignment for timespan properties.
- subsref SUBSREF Define field name indexing for timespan objects.
- timespan TIMESPAN timespan object class constructor.

SOURCE CODE ^

```
function pstr = char(ts ,varargin)
% CHAR convert a timespan object into a string.
%
%%%
```

pfq2ri

PURPOSE ^

Convert frequency/Q pole/zero representation into real

SYNOPSIS ^

function ri= pfq2ri(f0, Q)

DESCRIPTION ^

```
Convert frequency/Q pole/zero representation into real
and imaginary conjugate pairs. Returns [ri conj(ri)]

ri= pfq2ri(f0, Q)

M Hewitson 02-04-07


$Id: pfq2ri.m,v 1.10 2008/02/24 09:15:37 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ri= pfq2ri(f0, Q)

% Convert frequency/Q pole/zero representation into real
% and imaginary conjugate pairs. Returns [ri conj(ri)]
%
```

```
% ri= pfq2ri(f0, Q)
%
% M Hewitson 02-04-07
%
%
% $Id: pfq2ri.m,v 1.10 2008/02/24 09:15:37 hewitson Exp $
%

if(nargin==0)
  disp('usage: ri = fq2ri2(f0, Q)')
  return
elseif(nargin==1 || nargin==2 && isnan(Q))
    ri = -(1/(2*pi*f0));
elseif(nargin==2 && Q>=0)
    if Q < 0.5
      disp('Q < 0.5! Splitting to two real zeros;')
      ri = [-(1/(2*pi*f0)) -(1/(2*pi*f0))];
    else
      w0 = 2*pi*f0;
      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



% END

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for tsdata objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for tsdata objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for tsdata objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.15 2008/02/22 22:26:41 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for tsdata objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for tsdata objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.15 2008/02/22 22:26:41 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

char

PURPOSE ^

CHAR convert a param object into a string.

SYNOPSIS ^

function pstr = char(p, varargin)

DESCRIPTION ^

```
 CHAR convert a param object into a string.

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

 DESCRIPTION: CHAR convert a param object into a string.

 CALL:        string = char(tsdata)

 VERSION:     $Id: char.m,v 1.4 2008/02/13 11:39:35 mauro Exp $

 HISTORY:     04-02-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- get GET get tsdata properties.

This function is called by:

- char CHAR convert a param object into a string.
- display DISPLAY overloads display functionality for tsdata objects.
- get_xy_values GET_XY_VALUES returns the x-axis and/or y-axis values.
- isfield ISFIELD tests if the given field is one of the object properties.
- ne NE overloads the ~= operator for tsdata objects.
- save SAVE a tsdata object to file.
- set_xy_axis SET_XY_AXIS set the x-axis and y-axis values.
- single_operation SINGLE_OPERATION implements specified operator overload for tsdata objects.

- subsasgn SUBSASGN define index assignment for for tsdata properties.
- subsref SUBSREF Define field name indexing for tsdata objects.

SOURCE CODE ^

```
function pstr = char(p, varargin)
% CHAR convert a param object into a string.
%
%%%
```

subsref

PURPOSE ^

SUBSREF Define field name indexing for xydata objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for xydata objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for xydata objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.8 2008/02/22 22:26:40 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for xydata objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for xydata objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.8 2008/02/22 22:26:40 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

char

PURPOSE ^

CHAR convert a param object into a string.

SYNOPSIS ^

function pstr = char(p, varargin)

DESCRIPTION ^

```
 CHAR convert a param object into a string.

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

 DESCRIPTION: CHAR convert a param object into a string.

 CALL:        string = char(xydata)

 VERSION:     $Id: char.m,v 1.4 2008/02/13 12:57:47 mauro Exp $

 HISTORY:     04-02-2007 Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- get GET gets xydata properties.

This function is called by:

- char CHAR convert a param object into a string.
- display DISPLAY overloads display functionality for xydata objects.
- get_xy_values GET_XY_VALUES returns the x-axis and/or y-axis values.
- isfield ISFIELD tests if the given field is one of the object properties.
- ne NE overloads the ~= operator for xydata objects.
- save SAVE an xydata object to file.
- set_xy_axis SET_XY_AXIS set the x-axis and y-axis values.
- single_operation SINGLE_OPERATION implements specified operator overload for xydata objects.

- subsasgn SUBSASGN define index assignment for xydata properties.
- subsref SUBSREF Define field name indexing for xydata objects.

SOURCE CODE ^

```
function pstr = char(p, varargin)
% CHAR convert a param object into a string.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: CHAR convert a param object into a string.
%
% CALL:        string = char(xydata)
%
% VERSION:     $Id: char.m,v 1.4 2008/02/13 12:57:47 mauro Exp $
%
% HISTORY:     04-02-2007 Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION= '$Id: char.m,v 1.4 2008/02/13 12:57:47 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(p, 'xydata') && ischar(varargin{1})
    in = char(varargin{1});
    if strcmp(in, 'Params')
      pstr = plist();
      return
    elseif strcmp(in, 'Version')
      pstr = VERSION;
      return
    elseif strcmp(in, 'Category')
      pstr = CATEGORY;
      return
    end
  end
end

pstr = get(p,'name');


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

phase

PURPOSE ^

PHASE return the phase in degrees for a given complex input.

SYNOPSIS ^

function p = phase(resp)

DESCRIPTION ^

```
 PHASE return the phase in degrees for a given complex input.

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

 DESCRIPTION: PHASE return the phase in degrees for a given complex
              input. Supposed to be analogous to angle but return degrees
              instead of radians.

 COMMENT: This is just a wrapper to call ltpda_phase.

 CALL:       p = phase(resp)

 INPUTS:     resp - complex number

 OUTPUTS:    p    - phase

 VERSION: $Id: phase.m,v 1.1 2008/01/20 12:48:50 hewitson Exp $

 HISTORY: dd-mm-yyyy M Hewitson
            Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ltpda_phase LTPDA_PHASE return the phase in degrees for a given complex input.

This function is called by:

- ltpda_unwrapdeg LTPDA_UNWRAPDEG Unwrap a phase vector given in degrees.

SOURCE CODE ^

```
function p = phase(resp)
% PHASE return the phase in degrees for a given complex input.
%
%%%
```

subsref

PURPOSE ^

SUBSREF Define field name indexing for xyzdata objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for xyzdata objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for xyzdata objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.3 2008/02/22 22:26:40 hewitson Exp $

 HISTORY:     24-12-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for xyzdata objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for xyzdata objects.
%
% EXAMPLES:    All possible accesses are possible.
%
```

```
% VERSION:     $Id: subsref.m,v 1.3 2008/02/22 22:26:40 hewitson Exp $
%
% HISTORY:     24-12-07 M Hewitson
%                 Creation
%
%%%
```

logindlg

PURPOSE ^

LOGINDLG Dialog for visually secure login.

SYNOPSIS ^

function [varargout]=logindlg(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: logindlg.m,v 1.4 2008/02/22 10:58:34 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- mysql_connect MYSQL_CONNECT connects to an AO repository and returns the connection object.

SUBFUNCTIONS ^

- function KeyPress_Function(h,eventdata,fig)
- function Cancel(h,eventdata,fig)
- function OK(h,eventdata,fig)
- function Escape(h,eventdata,fig)

SOURCE CODE ^

```
function [varargout]=logindlg(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: logindlg.m,v 1.4 2008/02/22 10:58:34 hewitson Exp $

if nargin == 2
  Login    = varargin{1};
  password = varargin{2};
else
  Login = '';
  password = '';
end

for i=1:numel(password), passwordHidden(i)='*'; end
if ~exist('passwordHidden','var'), passwordHidden = ''; end

% Get Properties
Color = get(0,'DefaultUicontrolBackgroundcolor');

set(0,'Units','characters')
Screen = get(0,'screensize');
Position = [Screen(3)/2-17.5 Screen(4)/2-4.75 35 9.5];
set(0,'Units','pixels')


% Create the GUI
gui.main = dialog('HandleVisibility','on',...
    'IntegerHandle','off',...
    'Menubar','none',...
    'Name','LTPDA Repository Login',...
    'NumberTitle','off',...
    'Tag','logindlg',...
    'Color',Color,...
    'Units','characters',...
    'Userdata','logindlg',...
    'Position',Position);

    set(gui.main,'Closerequestfcn',{@Cancel,gui.main},'Keypressfcn',
{@Escape,gui.main})

% Texts
gui.login_text = uicontrol
(gui.
main,'Style','text','FontSize',8,'HorizontalAlign','center','Units','characters','String','Login','Position',
[1 7.65 33 1]);
gui.password_text = uicontrol
(gui.
```

```
main,'Style','text','FontSize',8,'HorizontalAlign','center','Units','characters','String','Password','Position',
[1 4.15 33 1]);
0066
0067 % Edits
0068 gui.edit1 = uicontrol
(gui.
main,'Style','edit','FontSize',8,'HorizontalAlign','center','BackgroundColor','white','Units','characters','String',
Login,'Position',[1 6.02 33 1.7]);
0069 gui.edit2 = uicontrol
(gui.
main,'Style','edit','FontSize',8,'HorizontalAlign','center','BackgroundColor','white','Units','characters','String',
passwordHidden,'Position',[1 2.52 33 1.7],'KeyPressfcn',{@KeyPress_Function,
gui.main},'Userdata',password);
0070
0071 % Buttons
0072 gui.OK = uicontrol
(gui.
main,'Style','push','FontSize',8,'Units','characters','String','OK','Position',
[2 .2 15 1.7],'Callback',{@OK,gui.main});
0073 gui.Cancel = uicontrol
(gui.
main,'Style','push','FontSize',8,'Units','characters','String','Cancel','Position',
[18 .2 15 1.7],'Callback',{@Cancel,gui.main});
0074
0075 setappdata(0,'logindlg',gui) % Save handle data
0076 setappdata(gui.main,'Check',0) % Error check setup. If Check remains 0 an
empty cell array will be returned
0077
0078 uicontrol(gui.edit1) % Make the first edit box active
0079
0080 % Pause the GUI and wait for a button to be pressed
0081 uiwait(gui.main)
0082
0083 Check = getappdata(gui.main,'Check'); % Check to see if a button was pressed
0084
0085 % Format output
0086 if Check == 1
0087     Login = get(gui.edit1,'String');
0088     Password = get(gui.edit2,'Userdata');
0089
0090     varargout(1) = {Login};
0091     varargout(2) = {Password};
0092 else % If OK wasn't pressed output nothing
0093         varargout(1) = {[]};
0094         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

%% 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)

%% 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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

subsref

PURPOSE ^

SUBSREF Define field name indexing for zero objects.

SYNOPSIS ^

function B = subsref(A, S)

DESCRIPTION ^

```
 SUBSREF Define field name indexing for zero objects.

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

 DESCRIPTION: SUBSREF Define field name indexing for zero objects.

 EXAMPLES:    All possible accesses are possible.

 VERSION:     $Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a zero object into a string.

This function is called by:

SOURCE CODE ^

```
function B = subsref(A, S)
% SUBSREF Define field name indexing for zero objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: SUBSREF Define field name indexing for zero objects.
%
```

```
% EXAMPLES:    All possible accesses are possible.
%
% VERSION:     $Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

collect_inputs

PURPOSE ^

COLLECT_INPUTS Collect the AOs and Plists.

SYNOPSIS ^

function [aos, pls, invars] = collect_inputs(varargin, in_names)

DESCRIPTION ^

```
 COLLECT_INPUTS Collect the AOs and Plists.

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

 DESCRIPTION: COLLECT_INPUTS Collect the AOs and Plists.

 CALL:        [aos, pls, invars] = collect_inputs(varargin, in_names);

 INPUTS:      varargin: A list of analysis objects (ao's) and
                        parameter lists (plist's)
              in_names: Corresponding variable names of the contents in
                        varargin

 OUTPUTS:     aos:    Collection of all analysis objects
              pls:    Collection of parameter lists
              invars: Collection of the ao names of the corresponding ao.

 VERSION:     $Id: collect_inputs.m,v 1.4 2008/02/25 19:46:01 hewitson Exp $

 HISTORY:     08-05-07 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function [aos, pls, invars] = collect_inputs(varargin, in_names)
% COLLECT_INPUTS Collect the AOs and Plists.
%
%%%
```

```
      if ~isempty(a.name) && ~strcmp(a.name, 'None')
        ao_name = a.name;
      end
      invars{end+1} = ao_name;
    else
      for ii=1:numel(a)
        if ~isempty(a(ii).name) && ~strcmp(a(ii).name, 'None')
          ao_name = a(ii).name;
        end
        [I,J] = ind2sub(size(a),ii);
        invars{end+1} = sprintf('%s(%d,%d)', ao_name, I, J);
      end
    end
  end
  if isa(varargin{j}, 'plist')
    pls = [pls varargin{j}];
  end
end

if isempty(pls)
  pls = plist;
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_allgrid

PURPOSE ^

LTPDA_ALLGRID Set all the grids to 'state'

SYNOPSIS ^

function ltpda_allgrid(state)

DESCRIPTION ^

```
 LTPDA_ALLGRID Set all the grids to 'state'

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

 DESCRIPTION: LTPDA_ALLGRID Set all the grids to 'state'

 CALL:        ltpda_allgrid('state')

 VERSION:     $Id: ltpda_allgrid.m,v 1.4 2007/07/16 12:52:20 ingo Exp $

 HISTORY:     17-11-2002 M Hewitson
                 Creation.

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_allgrid(state)
% LTPDA_ALLGRID Set all the grids to 'state'
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: LTPDA_ALLGRID Set all the grids to 'state'
%
% CALL:        ltpda_allgrid('state')
%
```

```
% VERSION:     $Id: ltpda_allgrid.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
%
% HISTORY:     17-11-2002 M Hewitson
%                 Creation.
%
%%%
```

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.

%%%
```

SUBFUNCTIONS ^

- function fcnname = getFcnName(name)
- function str = getNodeInputs(invars)

SOURCE CODE ^

```
function varargout = plot(varargin)
% PLOT plots a history object as a tree diagram.
%
%%%
```

```
  curr_axes = axes;
  if nargin >= 2
    varargin = varargin(2:end);
  else
    varargin = {};
  end

%% Used call: plot (curr_axes_handle, history, arg)
elseif ishandle(varargin{1})

  curr_axes = varargin{1};
  axes(curr_axes);
  hists = varargin{2};
  if nargin >= 3
    varargin = varargin(3:end);
  else
    varargin = {};
  end

else
  error (['### the first input should be a history object or an axes handle' ...
          sprintf(' varargin{1} = [%s]',varargin{1})]);
end

%% collect the parameter and put them into the parameter list
plot_args = {};
pl        = plist();
args      = varargin;

while ~isempty(args)

  if isa(args{1}, 'plist')
    arg  = args{1};
    pl   = append(pl, arg);
    args = args(2:end);

  % Add the key-values to the parameter list
  elseif length(args) >= 2
    if  ischar(args{1})
      arg = args{1};
      val = args{2};
      pl  = append(pl, param(arg, val));

      plot_args{end+1} = arg;
      plot_args{end+1} = val;
      args = args(3:end);

    else
      error('### the key [%s] is not from the type ''char''', char(arg));
    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),...
                  [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


%-----------------------------------------------------------------------------
% compute strings to display for function name
function fcnname = getFcnName(name)

switch name
  case 'mtimes'
    fcnname = '\times';
  case 'times'
```

```
    fcnname = '\times';
  case 'plus'
    fcnname = '+';
  case 'minus'
    fcnname = '-';
  case 'sqrt'
    fcnname = '\surd';
  case 'mrdivide'
    fcnname = '/';
  case 'rdivide'
    fcnname = '/';
  otherwise
    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


```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_allxaxis

PURPOSE ^

LTPDA_ALLXSCALE Set all the x scales on the current figure.

SYNOPSIS ^

function ltpda_allxaxis(x1, x2)

DESCRIPTION ^

```
 LTPDA_ALLXSCALE Set all the x scales on the current figure.

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

 DESCRIPTION: LTPDA_ALLXAXIS Set all the xaxis ranges on the current figure.

 CALL:        ltpda_allxaxis(x1, x2)

 VERSION:     $Id: ltpda_allxaxis.m,v 1.6 2007/07/30 16:04:09 ingo Exp $

 HISTORY:     17-11-2002 M Hewitson
                 Creation.

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_allxaxis(x1, x2)
% LTPDA_ALLXSCALE Set all the x scales on the current figure.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DESCRIPTION: LTPDA_ALLXAXIS Set all the xaxis ranges on the current figure.
%
% CALL:        ltpda_allxaxis(x1, x2)
%
```

```
% VERSION:     $Id: ltpda_allxaxis.m,v 1.6 2007/07/30 16:04:09 ingo Exp $
%
% HISTORY:     17-11-2002 M Hewitson
%                 Creation.
%
%%%
```

plot

PURPOSE ^

PLOT plots a specwin object.

SYNOPSIS ^

function varargout = plot(ws, varargin)

DESCRIPTION ^

```
 PLOT plots a specwin object.

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

 DESCRIPTION: PLOT plots a specwin object.

 CALL:            plot(specwin)
             h = plot(specwin)

 VERSION:     $Id: plot.m,v 1.6 2008/02/13 13:46:07 mauro Exp $

 HISTORY:     06-02-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a specwin object into a string.
- plot PLOT plots a specwin object.
- set SET sets a specwin property.

This function is called by:

- plot PLOT plots a specwin object.

SOURCE CODE ^

```
function varargout = plot(ws, varargin)
% PLOT plots a specwin object.
```

```
%
%%%
```

```
  ylabel('amplitude');
  titleStr = [titleStr, ltpda_label(w.name), ', '];
  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)];
end

legend(legendStr);
titleStr = titleStr(1:end-2);
title(sprintf('Window: %s', titleStr));

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

hold off

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

compute_interval

PURPOSE ^

COMPUTE_INTERVAL compute the interval of the time span.

SYNOPSIS ^

function ts = compute_interval(ts)

DESCRIPTION ^

```
 COMPUTE_INTERVAL compute the interval of the time span.

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

 DESCRIPTION: COMPUTE_INTERVAL compute the interval of the time span.

 CALL:        t1 = compute_interval(t1)

 VERSION:     $Id: compute_interval.m,v 1.3 2008/02/13 10:20:16 ingo Exp $

 HISTORY:     03-08-2007 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function ts = compute_interval(ts)
0002 % COMPUTE_INTERVAL compute the interval of the time span.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: COMPUTE_INTERVAL compute the interval of the time span.
0007 %
0008 % CALL:        t1 = compute_interval(t1)
0009 %
```

```
% VERSION:     $Id: compute_interval.m,v 1.3 2008/02/13 10:20:16 ingo Exp $
%
% HISTORY:     03-08-2007 Diepholz
%                 Creation
%
%%%
```

```
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_allxlabel

PURPOSE ^

LTPDA_ALLXLABEL Set all the x-axis labels on the current figure.

SYNOPSIS ^

function ltpda_allxlabel(label)

DESCRIPTION ^

```
 LTPDA_ALLXLABEL Set all the x-axis labels on the current figure.

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

 DESCRIPTION: LTPDA_ALLXLABEL Set all the x-axis labels on the current figure.

 CALL:        ltpda_allxlabel(label)

 VERSION:     $Id: ltpda_allxlabel.m,v 1.5 2007/07/30 16:04:09 ingo Exp $

 HISTORY:     17-11-2002 M Hewitson
                 Creation.

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_allxlabel(label)
% LTPDA_ALLXLABEL Set all the x-axis labels on the current figure.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: LTPDA_ALLXLABEL Set all the x-axis labels on the current figure.
%
% CALL:        ltpda_allxlabel(label)
%
```

```
% VERSION:     $Id: ltpda_allxlabel.m,v 1.5 2007/07/30 16:04:09 ingo Exp $
%
% HISTORY:     17-11-2002 M Hewitson
%                 Creation.
%
%%%
```

time

PURPOSE ^

TIME time object class constructor.

SYNOPSIS ^

function varargout = time(varargin)

DESCRIPTION ^

```
 TIME time object class constructor.

%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a time object into a string.
- set SET set time properties.

This function is called by:

- plus PLUS overloads - operator for time objects.
- set SET set time properties.
- subsasgn SUBSASGN define index assignment for time properties.

SUBFUNCTIONS ^

- function tt = init
- function fs = timeFromRepository(pl, VERSION, ALGONAME)
- function out = getDefaultPlist(varargin)

SOURCE CODE ^

```
function varargout = time(varargin)
% TIME time object class constructor.
%
%%%
```

```
%
% VERSION:      $Id: time.m,v 1.22 2008/02/24 10:15:56 hewitson Exp $
%
% HISTORY:      23-07-2007 Diepholz
%                  Creation
%
%%%
```

```
% Define a utc calendar
cal_utc    = GregorianCalendar(timezone);

% Get the unix time (timezone = UTC) in milliseconds
utc_epoch_milli = cal_utc.getTimeInMillis;

%%%%%%%%%%%%%%%%%%%%%%%%%   define time properties   %%%%%%%%%%%%%%%%%%%%%%%%%

  function tt = init
    tt.name            = 'time-object';
    tt.utc_epoch_milli = utc_epoch_milli;

    tt.timezone        = timezone;

    tt.timeformat      = timeformat(setup_format_str,   ...
                                    default_matlab_str, ...
                                    default_matlab_nr,  ...
                                    default_java_str);

    t_format = java.text.SimpleDateFormat(default_java_str);
    t_format.setTimeZone(tt.timezone);

    tt.time_str = char(t_format.format (tt.utc_epoch_milli));
    tt.created  = tt.time_str;
    tt.version  = VERSION;
    tt.plist    = '';
    tt = class(tt, 'time');

    % Set the ltpda_startup time format
    tt = set(tt, 'timeformat', tt.timeformat.format_str);
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%   Create time object   %%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%  t1 = time()   %%%%%%%%%%
if nargin == 0

  tt = init();

elseif nargin == 1

  %%%%%%%%%%   t1 = time(time-object)   %%%%%%%%%%
  if isa(varargin{1}, 'time')

    tt = varargin{1};

  %%%%%%%%%%   t1 = time('t0_default')   %%%%%%%%%%
  elseif strcmp(varargin{1}, 't0_default')

    tt = init();
    tt = set(tt, 'utc_epoch_milli', 0);

```

```
%%%%%%%%%%   t1 = time('2007-08-03 10:00:00')   %%%%%%%%%%
%%%%%%%%%%   t1 = time('foo.mat')               %%%%%%%%%%
%%%%%%%%%%   t1 = time('foo.xml')               %%%%%%%%%%
elseif ischar(varargin{1})

  % is this a filename?
  [path, name, ext, vers] = fileparts(varargin{1});
  if ~isempty(ext) && length(ext) == 4 && isempty(str2num(ext(2)))

    switch ext
      case '.mat'
        tt = load(varargin{1});
        tt = tt.a;
      case '.xml'
        root_node = xmlread(varargin{1});
        tt = ltpda_xmlread(root_node, 'time');
      otherwise
        error('### Unknown file type.');
    end

  else
    tt = init();
    tt = set(tt, 'time_str', varargin{1});
  end

%%%%%%%%%%   t1 = time(1186135200000)   %%%%%%%%%%
elseif isnumeric(varargin{1})

  tt = init();
  tt = set(tt, 'utc_epoch_milli', varargin{1});

%%%%%%%%%%   t1 = time(struct)   %%%%%%%%%%
elseif isstruct(varargin{1})

  tt = init();

  fields = fieldnames(varargin{1});
  for ii = 1:length(fields)
    field = fields{ii};
    %%% timeformat -> timeformat-object
    if strcmp(field, 'timeformat')
      tf = varargin{1}.timeformat;
      if isstruct(tf)
        tf = timeformat(tf);
      end
      tt.timeformat = tf;
    %% All other
    else
      try
        tt.(field) = varargin{1}.(field);
      catch
        error('### The field ''%s'' in the struct is not a time property.',
```

```
field)
        end
      end
    end

  %%%%%%%%%%   t1 = time(plist)   %%%%%%%%%%
  elseif isa(varargin{1}, 'plist')

    pl_timezone    = find(varargin{1}, 'timezone');
    pl_epoch_milli = find(varargin{1}, 'utc_epoch_milli');
    pl_time_str    = find(varargin{1}, 'time_str');
    pl_timeformat  = find(varargin{1}, 'timeformat');
    hostname       = find(varargin{1}, 'hostname');

    tt = init();

    if ~isempty(hostname)
      tt = timeFromRepository(varargin{1}, VERSION, ALGONAME);
    else

      if ~isempty(pl_timeformat)
        tt = set(tt, 'timeformat', pl_timeformat);
      else
        warning(['### The parameter list doesn''t contain a timeformat.'...
          sprintf('\n         ### Use the default timeformat %s'),...
          tt.timeformat.default_matlab_str]);
      end

      if ~isempty(pl_timezone)
        tt = set(tt, 'timezone', pl_timezone);
      else
        warning(['### The parameter list doesn''t contain a timezone. '...
          sprintf('\n         ### Use the default timezone ''UTC''')]);
        tt = set(tt, 'timezone', 'UTC');
      end

      if ~isempty(pl_epoch_milli)
        tt = set(tt, 'utc_epoch_milli', pl_epoch_milli);
      elseif ~isempty(pl_time_str)
        tt = set(tt, 'time_str', pl_time_str);

        if ~isempty(pl_timeformat)
          tt = set(tt, 'timeformat', pl_timeformat);
        end

      else
        error (['### The parameter list must contain '...
          '''utc_epoch_milli'' OR ''time_str''']])
      end
    end

    % Store input plist
```

```
    tt.plist = remove(varargin{1}, 'conn');

  %%%%%%%%%%   Unknown constructor   %%%%%%%%%%
  else
    error ('### Unknown constructor');
  end

elseif nargin == 2

  %%%%%%%%%%   t1 = time('14:00:00', 'HH:MM:SS')   %%%%%%%%%%
  if ischar(varargin{1}) && (ischar(varargin{2}) || isnumeric(varargin{2}))

    tt = init();
    tt = set(tt, 'time_str'  , varargin{1});
    tt = set(tt, 'timeformat', varargin{2});

  %%%%%%%%%%% From DATABASE
  elseif isa(varargin{1}, 'database')
    tt = retrieve(varargin{1}, varargin{2:end});
  %%%%%%%%%%   Unknown constructor   %%%%%%%%%%
  else
    error ('### Unknown constructor');
  end

else

  error ('### Unknown number of inputs');

end

varargout{1} = tt;



%-----------------------------------------------------------------------
% construct a time from a repository
%
  function fs = timeFromRepository(pl, VERSION, ALGONAME)

    dpl = getDefaultPlist('From Repository');
    pl  = combine(pl, dpl);

    % 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);
    fs  = [];
    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 a time
      if strcmp(tt, mfilename)
        %---- call database constructor
        a = ltpda_obj_retrieve(conn, id);
        %---- Add history
        %---- Add to output array
        fs = [fs a];
      else
        warning('    !skipping ID %d, type %s', id, tt);
      end

    end

    % close connection
    if closeConn
      close(conn);
    end

  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get default params
%
% HISTORY:     11-02-2008 M Hueller
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function out = getDefaultPlist(varargin)

    % list of available parameter sets
    sets = {'From Time Definition', 'From Repository'};

    if nargin == 0
```

```
      out = sets;
      return
    end

    set = varargin{1};
    switch set
      case 'From Repository'
        out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
      case 'From Time Definition'
        out = plist('utc_epoch_milli', 0, ...
                    'timezone', 'UTC', ...
                    'timeformat', 'yyyy-mm-dd HH:MM:SS.FFF');
      otherwise
        out = plist();
    end

  end

end % function varargout = time(varargin)
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_allxscale

PURPOSE ^

LTPDA_ALLXSCALE Set all the x scales on the current figure.

SYNOPSIS ^

function ltpda_allxscale(scale)

DESCRIPTION ^

```
 LTPDA_ALLXSCALE Set all the x scales on the current figure.

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

 DESCRIPTION: LTPDA_ALLXSCALE Set all the x scales on the current figure.

 CALL:        allxscale(scale)     scale = 'lin' or 'log';

 VERSION:     $Id: ltpda_allxscale.m,v 1.5 2007/07/30 16:04:09 ingo Exp $

 HISTORY:     02-05-2004 M Hewitson
                 Creation.

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_allxscale(scale)
% LTPDA_ALLXSCALE Set all the x scales on the current figure.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: LTPDA_ALLXSCALE Set all the x scales on the current figure.
%
% CALL:        allxscale(scale)     scale = 'lin' or 'log';
%
```

```
% VERSION:     $Id: ltpda_allxscale.m,v 1.5 2007/07/30 16:04:09 ingo Exp $
%
% HISTORY:     02-05-2004 M Hewitson
%                 Creation.
%
%%%
```

plus

PURPOSE ^

PLUS overloads - operator for time objects.

SYNOPSIS ^

function res = plus(t1,t2)

DESCRIPTION ^

```
 PLUS overloads - operator for time objects.

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

 DESCRIPTION: PLUS overloads - operator for analysis objects.
              This function adds a time in seconds to a time object.
              It is possible to define the addend as a string,
              number or a time object.

 CALL:        t2  = t1 + value (in seconds)
              sec = t1 + t2;

 EXAMPLES:    >> t1      = time('2007-07-01 12:23:33');
              >> t2      = t1 +  30;
              >> t3      = t1 + '30';
              >> seconds = t1 + t2;

 VERSION:     $Id: plus.m,v 1.7 2008/02/15 16:26:08 mauro Exp $

 HISTORY:     03-08-2007 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a time object into a string.
- format FORMAT Returns the time in the handover format.
- set SET set time properties.
- time TIME time object class constructor.

This function is called by:

SOURCE CODE ^

```
function res = plus(t1,t2)
% PLUS overloads - operator for time objects.
%
%%%
```

```
end

%%%%%   >> t2 = t1 +  30;   %%%%%
%%%%%   >> t2 = t1 + '30';   %%%%%
if ischar(t2)
  java_format_str_seconds = 'ss';

  utc_timezone   = java.util.TimeZone.getTimeZone('UTC');

  t_format_seconds = java.text.SimpleDateFormat(java_format_str_seconds);
  t_format_seconds.setTimeZone(utc_timezone);

  epoch_time_seconds = t_format_seconds.parse(t2);
  epoch_time_seconds = epoch_time_seconds.getTime;

  res = set(t1, 'utc_epoch_milli', t1.utc_epoch_milli+epoch_time_seconds);

%%%%%   >> number = t1 + t2;   %%%%%
elseif isa(t2, 'time')

  res = (t1-time(format(t1, 'yyyy-mm-dd'))) + (t2-time(format(t2, 'yyyy-mm-dd')));

%   res = (t1.utc_epoch_milli + t2.utc_epoch_milli)/1000;

else
  error ('### Unknown plus-operation to a time object.');
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

timeformat

PURPOSE ^

TIMEFORMAT timeformat object class constructor.

SYNOPSIS ^

function varargout = timeformat(varargin)

DESCRIPTION ^

```
 TIMEFORMAT timeformat object class constructor.

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

 DESCRIPTION:  TIMEFORMAT timeformat object class constructor.
               Create a timeformat object.

 PROPERTIES:   format_str         - current matlab string
               format_nr          - current matlab number
               default_matlab_str - default matlab string (read only)
               default_matlab_nr  - default matlab number (read only)
               default_java_str   - default java string   (read only)

 REMARK:       This class is a PRIVATE class and should not be uses by the
               user.
               The default properties must match together. They must represent
               the same time format.

 CONSTRUCTORS: timeformat(format_str,
                          default_matlab_str,
                          default_matlab_nr,
                          default_java_str)

 VERSION:      $Id: timeformat.m,v 1.8 2008/02/13 16:54:14 mauro Exp $

 HISTORY:      23-07-2007 Diepholz
                  Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a timeformat object into a string.

This function is called by:

- subsasgn SUBSASGN define index assignment for timeformat properties.

SUBFUNCTIONS ^

- function tf = init()

SOURCE CODE ^

```
function varargout = timeformat(varargin)
% TIMEFORMAT timeformat object class constructor.
%
%%%
```

```
    if strcmp(in, 'Params')
      varargout{1} = plist;
      return
    elseif strcmp(in, 'Version')
      varargout{1} = VERSION;
      return
    elseif strcmp(in, 'Category')
      varargout{1} = CATEGORY;
      return
    end
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%   define time properties   %%%%%%%%%%%%%%%%%%%%%%%%

  function tf = init()
    tf.format_str = '';
    tf.format_nr  = -1;
    tf.default_matlab_str = '';
    tf.default_matlab_nr  = -1;
    tf.default_java_str   = '';
    tf = class(tf, 'timeformat');
  end


%%%%%%%%%%%%%%%%%%%%%%%%   Create timeformat object   %%%%%%%%%%%%%%%%%%%%%%%%

% Create empty time format object
if nargin == 0
  tf = init();
  varargout{1} = tf;
elseif nargin == 1

  %%%%%%%%%%   tf = timeformat(timeformat-object)   %%%%%%%%%%
  if isa(varargin{1}, 'timeformat')

    tf = varargin{1};
    varargout{1} = tf;

  %%%%%%%%%%   tf = timeformat('foo.mat')   %%%%%%%%%%
  %%%%%%%%%%   tf = timeformat('foo.xml')   %%%%%%%%%%
  elseif ischar(varargin{1})

    filename = varargin{1};
    [path, name, ext, vers] = fileparts(filename);
    switch ext
      case '.mat'
        tf = load(filename);
        tf = tf.a;
      case '.xml'
        root_node = xmlread(filename);
        tf = ltpda_xmlread(root_node, 'timeformat');
```

```
      otherwise
        error('### Unknown file type.');
    end

    varargout{1} = tf;

  %%%%%%%%%%   t1 = time(struct)   %%%%%%%%%%
  elseif isstruct(varargin{1})

    tf = init();

    fields = fieldnames(varargin{1});
    for ii = 1:length(fields)
      field = fields{ii};
      try
        tf.(field) = varargin{1}.(field);
      catch
        error('### The field ''%s'' in the struct is not a timeformat property.', field)
      end
    end
  else
    error ('### Unknown constructor');
  end

  varargout{1} = tf;

elseif nargin == 2
  %%%%%%%%%%% From DATABASE
  if isa(varargin{1}, 'database')
    varargout{1} = retrieve(varargin{1}, varargin{2:end});
  else
    error('### Incorrect constructor call.');
  end

elseif nargin == 4
  tf = init();

  tf.format_str         = varargin{1};
  tf.default_matlab_str = varargin{2};
  tf.default_matlab_nr  = varargin{3};
  tf.default_java_str   = varargin{4};

  if strcmp(tf.format_str, tf.default_matlab_str)
    tf.format_nr = tf.default_matlab_nr;
  end

  varargout{1} = tf;
end
```

```
0142 end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

conv_noisegen

PURPOSE ^

CONV_NOISEGEN calls the matlab function conv.m to convolute poles and

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 1.5 2008/02/22 11:12:46 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function [b,a] = conv_noisegen(pol,zer)
0002 % 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 1.5 2008/02/22 11:12:46 hewitson Exp $
%

[m,k] = size(pol);
[n,l] = size(zer);

coefb = pol(1,:);

for i = 2:m
    coefb = conv(coefb, pol(i,:));
end

b = nonzeros(coefb);

if n~=0
    coefa = zer(1,:);
    for i = 2:n
        coefa = conv(coefa, zer(i,:));
    end
    a = nonzeros(coefa);
else
    a = 1;
end

%normalize to bn = 1
m = length(b);
normfac = b(m);
b = b/normfac;
a = a/(normfac*sqrt(2));


```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_allyaxis

PURPOSE ^

LTPDA_ALLYAXIS Set all the yaxis ranges on the current figure.

SYNOPSIS ^

function ltpda_allyaxis(y1, y2)

DESCRIPTION ^

```
 LTPDA_ALLYAXIS Set all the yaxis ranges on the current figure.

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

 DESCRIPTION: LTPDA_ALLYAXIS Set all the yaxis ranges on the current figure.

 CALL:        allyaxis(y1, y2)

 VERSION:     $Id: ltpda_allyaxis.m,v 1.5 2007/07/30 16:04:09 ingo Exp $

 HISTORY:     17-11-2002 M Hewitson
                 Creation.

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_allyaxis(y1, y2)
% LTPDA_ALLYAXIS Set all the yaxis ranges on the current figure.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: LTPDA_ALLYAXIS Set all the yaxis ranges on the current figure.
%
% CALL:        allyaxis(y1, y2)
%
```

```
% VERSION:     $Id: ltpda_allyaxis.m,v 1.5 2007/07/30 16:04:09 ingo Exp $
%
% HISTORY:     17-11-2002 M Hewitson
%                 Creation.
%
%%%
```

ltpda_allylabel

PURPOSE ^

LTPDA_ALLYLABEL Set all the y-axis labels on the current figure.

SYNOPSIS ^

function ltpda_allylabel(label)

DESCRIPTION ^

```
 LTPDA_ALLYLABEL Set all the y-axis labels on the current figure.

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

 DESCRIPTION: LTPDA_ALLYLABEL Set all the y-axis labels on the current figure.

 CALL:        allylabel(label)

 VERSION:     $Id: ltpda_allylabel.m,v 1.5 2007/07/30 16:04:09 ingo Exp $

 HISTORY:     17-11-2002 M Hewitson
                 Creation.

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_allylabel(label)
% LTPDA_ALLYLABEL Set all the y-axis labels on the current figure.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: LTPDA_ALLYLABEL Set all the y-axis labels on the current figure.
%
% CALL:        allylabel(label)
%
```

```
% VERSION:     $Id: ltpda_allylabel.m,v 1.5 2007/07/30 16:04:09 ingo Exp $
%
% HISTORY:     17-11-2002 M Hewitson
%                 Creation.
%
%%%
```

polydetrend

PURPOSE ^

POLYDETREND detrends the input data vector with a polynomial.

SYNOPSIS ^

function y = polydetrend(varargin)

DESCRIPTION ^

```
 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: polydetrend.m,v 1.2 2007/07/16 12:52:21 ingo Exp $

 HISTORY:     30-05-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpda_polydetrend LTPDA_POLYDETREND detrends the input analysis object using a polynomial of degree N.

SOURCE CODE ^

```
function y = polydetrend(varargin)
% POLYDETREND detrends the input data vector with a polynomial.
%
%%%
```

cp2iir

PURPOSE ^

CP2IIR Return a,b IIR filter coefficients for a complex pole designed using the bilinear transform.

SYNOPSIS ^

function [a,b] = cp2iir(p, fs)

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 1.5 2008/02/12 21:38:23 hewitson Exp $

 HISTORY:     03-04-2007 M Hewitson
                  Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- get GET get pole properties.

This function is called by:

SOURCE CODE ^

```
function [a,b] = cp2iir(p, fs)
% CP2IIR Return a,b IIR filter coefficients for a complex pole designed using the bilinear transform.
%
%%%
```

```
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;

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_allyscale

PURPOSE ^

LTPDA_ALLYSCALE Set all the Y scales on the current figure.

SYNOPSIS ^

function ltpda_allyscale(scale)

DESCRIPTION ^

```
 LTPDA_ALLYSCALE Set all the Y scales on the current figure.

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

 DESCRIPTION: LTPDA_ALLYSCALE Set all the Y scales on the current figure.

 CALL:        allyscale(scale)         scale = 'lin' or 'log';

 VERSION:     $Id: ltpda_allyscale.m,v 1.5 2007/07/30 16:04:09 ingo Exp $

 HISTORY:     02-05-2004 M Hewitson
                 Creation.

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_allyscale(scale)
% LTPDA_ALLYSCALE Set all the Y scales on the current figure.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: LTPDA_ALLYSCALE Set all the Y scales on the current figure.
%
% CALL:        allyscale(scale)         scale = 'lin' or 'log';
%
```

```
% VERSION:     $Id: ltpda_allyscale.m,v 1.5 2007/07/30 16:04:09 ingo Exp $
%
% HISTORY:     02-05-2004 M Hewitson
%                 Creation.
%
%%%
```

timespan

PURPOSE ^

TIMESPAN timespan object class constructor.

SYNOPSIS ^

function varargout = timespan(varargin)

DESCRIPTION ^

```
 TIMESPAN timespan object class constructor.

%%%
```

This function calls:

- char CHAR convert a timespan object into a string.
- set SET set timespan properties.

This function is called by:

- subsasgn SUBSASGN define index assignment for timespan properties.

SUBFUNCTIONS ^

- function t_span = init
- function fs = timespanFromRepository(pl, VERSION, ALGONAME)
- function out = getDefaultPlist(varargin)

SOURCE CODE ^

```
function varargout = timespan(varargin)
% TIMESPAN timespan object class constructor.
%
%%%
```

```
ALGONAME = mfilename;
VERSION  = '$Id: timespan.m,v 1.17 2008/02/24 10:15:55 hewitson Exp $';
CATEGORY = 'Constructor';

% Check if this is a call for parameters
if (nargin == 2 || nargin == 3)
  if isa(varargin{1}, 'timespan') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      if nargin == 2
        varargout{1} = getDefaultPlist();
      else
        varargout{1} = getDefaultPlist(varargin{3});
      end
      return
    elseif strcmp(in, 'Version')
      varargout{1} = VERSION;
      return
    elseif strcmp(in, 'Category')
      varargout{1} = CATEGORY;
      return
    end
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%   define time properties   %%%%%%%%%%%%%%%%%%%%%%%%

  function t_span = init
    t_span.name       = 'time span-object';
    t_span.start      = time();
    t_span.end        = time();
    t_span.timeformat = t_span.start.timeformat;
    t_span.interval   = '';
    t_span.timezone   = t_span.start.timezone;
    t_span.created    = time();
    t_span.version    = VERSION;
    t_span.plist      = plist;
    t_span = class (t_span, 'timespan');
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%   Create time object   %%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%  t_span = timespan()   %%%%%%%%%%
if nargin == 0

  t_span = init;

elseif nargin == 1

  %%%%%%%%%%   t1 = time(time-object)   %%%%%%%%%%
  if isa(varargin{1}, 'timespan')
```

```

    t_span = varargin{1};

  %%%%%%%%%%   tf = timespan('foo.mat')   %%%%%%%%%%
  %%%%%%%%%%   tf = timespan('foo.xml')   %%%%%%%%%%
  elseif ischar(varargin{1})

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

    varargout{1} = t_span;

  %%%%%%%%%%   t1 = time(plist)   %%%%%%%%%%
  elseif isa(varargin{1}, 'plist')

    pl_start      = find(varargin{1}, 'start');
    pl_end        = find(varargin{1}, 'end');
    pl_timezone   = find(varargin{1}, 'timezone');
    pl_timeformat = find(varargin{1}, 'timeformat');
    hostname      = find(varargin{1}, 'hostname');

    if ~isempty(hostname)
      t_span = timespanFromRepository(varargin{1}, VERSION, ALGONAME);
    else
      if isempty(pl_start) || isempty(pl_end)
        error('### The plist must contain the keys: ''start'' and ''end''');
      end

      t_span = init();

      if ~isempty(pl_timeformat)
        format_nr = str2double(pl_timeformat);
        if isnumeric(format_nr) && ~isnan(format_nr)
          t_span = set(t_span, 'timeformat', format_nr);
        else
          t_span = set(t_span, 'timeformat', pl_timeformat);
        end
      end

      if ~isempty(pl_timezone)
        t_span = set(t_span, 'timezone', pl_timezone);
      end
```

```

      t_span = set(t_span, 'start', pl_start);
      t_span = set(t_span, 'end', pl_end);
    end

    % Store input plist
    t_span.plist = remove(varargin{1}, 'conn');

  %%%%%%%%%%   t1 = time(struct)   %%%%%%%%%%
  elseif isstruct(varargin{1})

    t_span = init();

    fields = fieldnames(varargin{1});
    for ii = 1:length(fields)
      field = fields{ii};
      %%% start -> time-object
      if strcmp(field, 'start')
        start_t = varargin{1}.start;
        if isstruct(start_t)
          start_t = time(start_t);
        end
        t_span.start = start_t;
      %%% end -> time-object
      elseif strcmp(field, 'end')
        end_t = varargin{1}.end;
        if isstruct(end_t)
          end_t = time(end_t);
        end
        t_span.end = end_t;
      %%% timeformat -> timeformat-object
      elseif strcmp(field, 'timeformat')
        tf = varargin{1}.timeformat;
        if isstruct(tf)
          tf = timeformat(tf);
        end
        t_span.timeformat = tf;
      %%% All other
      else
        try
          t_span.(field) = varargin{1}.(field);
        catch
          error('### The field ''%s'' in the struct is not a timespan property.', field)
        end
      end
    end

    varargout{1} = t_span;

  else
    error ('### Unknown constructor');
```

```
  end

elseif nargin == 2

  %%%%%%%%%%  t_span = timespan('14:00:00', '14:00:05')   %%%%%%%%%%
  if ischar(varargin{1}) && ischar(varargin{2})

    t_span = init;
    t_span.start = time(varargin{1});
    t_span.end   = time(varargin{2});

    % Adjust the 'timeformat' AND the 'timezone' of the end time to the start time.
    t_span = set(t_span, 'timeformat', t_span.start.timeformat);
    t_span = set(t_span, 'timezone',   t_span.start.timezone);

    % Compute the interval
    t_span = compute_interval(t_span);

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

    t_span = retrieve(varargin{1}, varargin{2:end});

  %%%%%%%%%%  t_span = timespan(time, time)   %%%%%%%%%%
  elseif isa(varargin{1}, 'time') && isa(varargin{2}, 'time')

    t_span = init;
    t_span.start = varargin{1};
    t_span.end   = varargin{2};

    % Adjust the 'timeformat' AND the 'timezone' of the end time to the start time.
    t_span = set(t_span, 'timeformat', t_span.start.timeformat);
    t_span = set(t_span, 'timezone',   t_span.start.timezone);

    % Compute the interval
    t_span = compute_interval(t_span);

  %%%%%%%%%%  t_span = timespan(time, '14:00:05')   %%%%%%%%%%
  elseif isa(varargin{1}, 'time') && ischar(varargin{2})

    t_span = init;
    t_span.start = varargin{1};
    t_span.end   = time(varargin{2}, char(t_span.start.timeformat));

    % Adjust the 'timeformat' AND the 'timezone' of the end time to the start time.
    t_span = set(t_span, 'timeformat', t_span.start.timeformat);
    t_span = set(t_span, 'timezone',   t_span.start.timezone);

    % Compute the interval
```

```
    t_span = compute_interval(t_span);

  %%%%%%%%%%  t_span = timespan('14:00:00', time)   %%%%%%%%%%
  elseif ischar(varargin{1}) && isa(varargin{2}, 'time')

    t_span = init;
    t_span.end   = varargin{2};
    t_span.start = time(varargin{1}, char(t_span.end.timeformat));

    % Adjust the 'timeformat' AND the 'timezone' of the end time to the start time.
    t_span = set(t_span, 'timeformat', t_span.start.timeformat);
    t_span = set(t_span, 'timezone',   t_span.start.timezone);

    % Compute the interval
    t_span = compute_interval(t_span);

  else
    error ('  ### Unknown constructor.');

  end

elseif nargin == 3

  %%%%%%%%%%  t_span = timespan('14:00:00', '14:00:05', 'HH:MM:SS')   %%%%%%%%%%
  if ischar(varargin{1}) && ischar(varargin{2}) && ischar(varargin{3})

    t_span = init;
    t_span.start = time(varargin{1}, varargin{3});
    t_span.end   = time(varargin{2}, varargin{3});
    t_span.timeformat = t_span.start.timeformat;

    % Compute the interval
    t_span = compute_interval(t_span);

  elseif isa(varargin{1}, 'time') && isa(varargin{2}, 'time') && isa(varargin{3}, 'timeformat')

    t_span = init;
    t_span.start      = varargin{1};
    t_span.end        = varargin{2};
    t_span.timeformat = varargin{3};

    t_span.start = set(t_span.start, 'timeformat', t_span.timeformat);
    t_span.end   = set(t_span.end,   'timeformat', t_span.timeformat);

    % Compute the interval
    t_span = compute_interval(t_span);

  else
    error ('  ### Unknown constructor.');

```

```
  end

else

  error ('### Unknown number of inputs');

end


if t_span.start.utc_epoch_milli > t_span.end.utc_epoch_milli
  error('### The start time is larger than the end time.');
end

varargout{1} = t_span;

%-----------------------------------------------------------------------
% construct a timespan from a repository
%
  function fs = timespanFromRepository(pl, VERSION, ALGONAME)

    dpl = getDefaultPlist('From Repository');
    pl  = combine(pl, dpl);

    % 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);
    fs  = [];
    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 a timespan
      if strcmp(tt, mfilename)
```

```
        %---- call database constructor
        a = ltpda_obj_retrieve(conn, id);
        %---- Add history
        %---- Add to output array
        fs = [fs a];
      else
        warning('    !skipping ID %d, type %s', id, tt);
      end

    end

    % close connection
    if closeConn
      close(conn);
    end

  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get default params
%
% HISTORY:     11-02-2008 M Hueller
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function out = getDefaultPlist(varargin)

    % list of available parameter sets
    sets = {'From Timespan Definition', 'From Repository'};

    if nargin == 0
      out = sets;
      return
    end

    set = varargin{1};
    switch set
      case 'From Repository'
        out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
      case 'From Timespan Definition'
        out = plist('start', '1970-01-01 00:30:00.000', ...
                    'end', '1980-01-01 12:00:00.010', ....
                    'timezone', 'UTC', ...
                    'timeformat', 'yyyy-mm-dd HH:MM:SS.FFF');
      otherwise
        out = plist();
    end

  end
```

```
0393
0394
0395 end % varargout = timespan(varargin)
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_arma_freq

PURPOSE ^

LTPDA_ARMA_FREQ estimates the ARMA parameters of the transfer function

SYNOPSIS ^

function varargout = ltpda_arma_freq(varargin)

DESCRIPTION ^

```
 LTPDA_ARMA_FREQ estimates the ARMA parameters of the transfer function
 relating an output to an input

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

 DESCRIPTION: LTPDA_ARMA_FREQ estimates the ARMA parameters of the
 transfer function relating an output to an input time series. The
 algorithm uses the IV method to find a set of initial parameters
 and then performs an iterative search based on the Optimization
 Toolbox.

 CALL:        b = ltpda_arma_freq(ax,ay,pl)

 INPUTS:      ax  - analysis object containig the input time series
              ay  - analysis object containig the output time series
              pl  - parameters list

 OUTPUTS:     b  - cdata type analysis object containing the ARMA
                   parameters and the residual of the fit

 PARAMETERS:
  MaxIter     -  Maximum number of iterations to be performed by the
                 iterative search (default 4e2)
  MaxFunEvals -  Maximum number of function evaluations (default 1e2)
  TolX        -  Tolerance on the estimated parameter  (default 1e-6)
  TolFun      -  Tolerance on the evaluated function   (default 1e-6)
  UpBound     -  Array of parameters upper bounds for the iterative search
                 (default is 1 for each parameter)
  LowBound    -  Array of parameters lower bounds for the iterative search
                 (default is 1 for each parameter)
  ARMANum     -  MA order of the ARMA filter (default 2)
  ARMADen     -  AR order of the ARMA filter (default 1)

 VERSION:     $Id: ltpda_arma_freq.m,v 1.2 2008/02/25 17:46:50 miquel Exp $
```

```
 HISTORY:     25-02-2008 M Nofrarias
                 Creation


 TODO: - Add parameters errors
       - Pass from univariate to multivariate


%%%
```

```
%  TolFun      -  Tolerance on the evaluated function   (default 1e-6)
%  UpBound     -  Array of parameters upper bounds for the iterative search
%                 (default is 1 for each parameter)
%  LowBound    -  Array of parameters lower bounds for the iterative search
%                 (default is 1 for each parameter)
%  ARMANum     -  MA order of the ARMA filter (default 2)
%  ARMADen     -  AR order of the ARMA filter (default 1)
%
% VERSION:     $Id: ltpda_arma_freq.m,v 1.2 2008/02/25 17:46:50 miquel Exp $
%
% HISTORY:     25-02-2008 M Nofrarias
%                 Creation
%
%
% TODO: - Add parameters errors
%       - Pass from univariate to multivariate
%
%
%%%
```

```
    end
    if isa(varargin{j}, 'plist')
      ps = [ps varargin{j}];
    end
end


%% Check plist
if isempty(ps)
  pl = getDefaultPL();
else
  pl = combine(ps, getDefaultPL);
end


%% Fourier transform input varialbles

Fs=0.65;
L=length(xin)
NFFT = 2^nextpow2(L); % Next power of 2 from length of y
NFFT=length(xin);

x_ = fft(xin,NFFT)/L;
y_ = fft(yout',NFFT)/L;

Px = 2*abs(x_(1:NFFT/2)); % single-sided amplitude spectrum.
Py = 2*abs(y_(1:NFFT/2)); % single-sided amplitude spectrum.

f = Fs/2*linspace(0,1,NFFT/2)';



%% Iterative search using the Optimization Toolbox


% Options for the lsqcurvefit function
opt = optimset(...
'MaxIter',find(pl, 'MaxIter'),...
'TolX',find(pl, 'TolXm'),...
'TolFun',find(pl, 'TolFun'),...
'MaxFunEvals',find(pl, 'MaxFunEvals'),...
'Display','off');

% Upper and Lower Bounds for the parameters
ub=find(pl,'UpBound');
lb=find(pl,'LowBound');
if isempty(ub)
ub=ones(p+q,1);
pl = append(pl, param('UpBound', ub));
disp('! Parameters Upper Bound set to 1')
end
```

```
if isempty(lb)
lb=-ones(p+q,1);
pl = append(pl, param('LowBound', lb));
disp('! Parameters Lower Bound set to -1')
end

% Call to the Optimization Toolbox function
[par,res]=lsqcurvefit(@(par,xdata)filter([par(1:p)],[1 par(p+1:p+q)],xdata),...
       par0,x,y,lb,ub,opt);


%% Build output ao

% New output history
h = history(ALGONAME, VERSION, pl,[as(1).hist as(2).hist]);
h = set(h,'invars', invars);

% Make output analysis object
b = ao([par; res]);

% Name, mfilename, description for this object
b = setnh(b,...
    'name', ['ARMA param. Input: ' sprintf('%s ', char(invars{1})) ' Output: '
char(invars{2})],...
    'mfilename', ALGONAME, ...
    'description', find(pl,'description'),...
    'hist', h);

varargout{1} = b;


%% --- FUNCTIONS ---


% Get default parameters
function plo = getDefaultPL();
disp('* creating default plist...');
plo = plist();
plo = append(plo, param('MaxIter',4e2));
plo = append(plo, param('MaxFunEvals',1e2));
plo = append(plo, param('TolX',1e-6));
plo = append(plo, param('TolFun',1e-6));
plo = append(plo, param('ARMANum',2));
plo = append(plo, param('ARMADen',1));
plo = append(plo, param('fs', 1));
disp('* done.');



% siz=length(Px);
% % siz=200;
%
```

```
% xdataf=Px(1:siz)*1e5;
% ydataf=Py(1:siz)*1e5;
% % xdataf=Px.data.y;
% % ydataf=Py.data.y;
% global freq
% % freq=Px.data.x;
% freq =f(1:siz);


tic;
[x1f,resnorm1f]=...
    lsqcurvefit(@myfunf,p0,xdataf,ydataf,[-1 -1 -1],[1 1 1],options);
elaps=toc

% %
% %%
% % Ff = ((x1f(1)+x1f(2)*exp(-i*2*pi*f/0.65))./(1+x1f(3)*exp(-i*2*pi*f/0.65)));
% % F = ((x1(1)+x1(2)*exp(-i*2*pi*f/0.65))./(1+x1(3)*exp(-i*2*pi*f/0.65)));
% %
% % figure(1)
% % hold off
% % loglog(Px.data.x,ydataf./xdataf)
% % hold
% % loglog(Px.data.x,abs(Ff),'k')
% % loglog(Px.data.x,abs(F),'m')
% % loglog(f,abs((39.6e-3-39.5e-3*exp(-i*2*pi*f/0.65))./(1-0.996*exp(-i*2*pi*f/0.65))),'r')
% %
% %
%
% %%
%
% Fs=18;
% Lw=2;
%
% Ff = ((x1f(1)+x1f(2)*exp(-i*2*pi*f/0.65))./(1+x1f(3)*exp(-i*2*pi*f/0.65)));
% F = ((x1(1)+x1(2)*exp(-i*2*pi*f/0.65))./(1+x1(3)*exp(-i*2*pi*f/0.65)));
%
% figure(1)
% set(gca,'Fontsize',Fs)
% hold off
% loglog(f,Py./Px,'LineWidth',Lw)
% hold
% % % loglog(f,ydataf,'m')
% loglog(f,abs(Ff),'k','LineWidth',Lw)
% loglog(f,abs(F),'m','LineWidth',Lw)
%
% % loglog(f,(x1(1)+x1(2)*exp(-i*2*pi*f/1.32))./(1+x1(3)*exp(-i*2*pi*f/1.32)),'r')
% loglog(f,abs((39.6e-3-39.5e-3*exp(-i*2*pi*f/0.65))./(1-0.996*exp(-i*2*pi*f/0.65))),'r')
%
```

```
% legend('Empirical','Freq.domain','Time domain','Analytic')
%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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]

 VERSION:     $Id: tomfir.m,v 1.3 2008/02/12 20:07:37 hewitson Exp $

 HISTORY:     16-08-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pzmodel object into a string.
- resp RESP returns the complex response of a pzmodel as an Analysis Object.
- set SET sets a pole/zero model property.

This function is called by:

SOURCE CODE ^

```
function varargout = tomfir(varargin)
% 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]
%
% VERSION:     $Id: tomfir.m,v 1.3 2008/02/12 20:07:37 hewitson Exp $
%
% HISTORY:     16-08-07 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: tomfir.m,v 1.3 2008/02/12 20:07:37 hewitson Exp $';
CATEGORY = 'Operator';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'pzmodel') && ischar(varargin{2})
    in = char(varargin{2});
    if strcmp(in, 'Params')
      varargout{1} = plist;
      return
```

```
    elseif strcmp(in, 'Version')
      varargout{1} = VERSION;
      return
    elseif strcmp(in, 'Category')
      varargout{1} = CATEGORY;
      return
    end
  end
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');

% if isempty(nf)
%   nf = 1000;
% end
% % compute response of model
% if isempty(f2)
%   r = resp(pzm, plist('f1', 0, 'nf', nf));
% else
%   r = resp(pzm, plist('f1', 0, 'f2', f2, 'nf', nf));
% end

r = resp(pzm, plist('f1', 0, 'f2', fs/2, 'nf', 1000));

% Set fs
r = set(r, 'fs', fs);

% compute filter
f = mfir(r);

if nargout == 1
  varargout{1} = f;
else
  error('### Incorrect outputs.');
end


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

cz2iir

PURPOSE ^

CZ2IIR return a,b IIR filter coefficients for a complex zero designed using the bilinear transform.

SYNOPSIS ^

function [a,b] = cz2iir(z, fs)

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 1.5 2008/02/12 19:54:16 hewitson Exp $

 HISTORY:     03-04-2007 Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- get GET get zero properties.

This function is called by:

SOURCE CODE ^

```
function [a,b] = cz2iir(z, fs)
% CZ2IIR return a,b IIR filter coefficients for a complex zero designed using the bilinear transform.
%
%%%
```

```
a(3) = (-q*w02/2 - 2*q*fs*fs + w0*fs) / (q*w02);

b(1) =  1;
b(2) = -2;
b(3) = -1;

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_arma_time

PURPOSE ^

LTPDA_ARMA_TIME estimates the ARMA parameters of the transfer function

SYNOPSIS ^

function varargout = ltpda_arma_time(varargin)

DESCRIPTION ^

```
 LTPDA_ARMA_TIME estimates the ARMA parameters of the transfer function
 relating an output to an input

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

 DESCRIPTION: LTPDA_ARMA_TIME estimates the ARMA parameters of the
 transfer function relating an output to an input time series. The
 algorithm uses the IV method to find a set of initial parameters
 and then performs an iterative search based on the Optimization
 Toolbox.

 CALL:        b = ltpda_arma_time(ax,ay,pl)

 INPUTS:      ax  - analysis object containig the input time series
              ay  - analysis object containig the output time series
              pl  - parameters list

 OUTPUTS:     b  - cdata type analysis object containing the ARMA
                   parameters and the residual of the fit

 PARAMETERS:
  MaxIter     -  Maximum number of iterations to be performed by the
                 iterative search (default 4e2)
  MaxFunEvals -  Maximum number of function evaluations (default 1e2)
  TolX        -  Tolerance on the estimated parameter  (default 1e-6)
  TolFun      -  Tolerance on the evaluated function   (default 1e-6)
  UpBound     -  Array of parameters upper bounds for the iterative search
                 (default is 1 for each parameter)
  LowBound    -  Array of parameters lower bounds for the iterative search
                 (default is 1 for each parameter)
  ARMANum     -  MA order of the ARMA filter (default 2)
  ARMADen     -  AR order of the ARMA filter (default 1)

 VERSION:     $Id: ltpda_arma_time.m,v 1.1 2008/02/15 14:45:28 miquel Exp $
```

```
 HISTORY:     15-02-2008 M Nofrarias
                 Creation


 TODO: - Add parameters errors
       - Pass from univariate to multivariate


%%%
```

```
%  TolFun      -  Tolerance on the evaluated function   (default 1e-6)
%  UpBound     -  Array of parameters upper bounds for the iterative search
%                 (default is 1 for each parameter)
%  LowBound    -  Array of parameters lower bounds for the iterative search
%                 (default is 1 for each parameter)
%  ARMANum     -  MA order of the ARMA filter (default 2)
%  ARMADen     -  AR order of the ARMA filter (default 1)
%
% VERSION:     $Id: ltpda_arma_time.m,v 1.1 2008/02/15 14:45:28 miquel Exp $
%
% HISTORY:     15-02-2008 M Nofrarias
%                 Creation
%
%
% TODO: - Add parameters errors
%       - Pass from univariate to multivariate
%
%
%%%
```

```
    end
    if isa(varargin{j}, 'plist')
      ps = [ps varargin{j}];
    end
end


%% Check plist
if isempty(ps)
  pl = getDefaultPL();
else
  pl = combine(ps, getDefaultPL);
end


%% Initialise variables
x=as(1).data.y;
y=as(2).data.y;
p = find(pl, 'ARMANum');
q = find(pl, 'ARMADen');
disp(sprintf('! ARMA orders p = %d, q = %d', p,q))

%% First estimate through Instrument Variables (IV) Method

for i=1:p
     inst1(:,i) = [zeros(i,1); x(1:length(x)-i)];
end
for i=1:q
     inst2(:,i) = [zeros(i,1); -y(1:length(y)-i)];
end

inst = [inst1 inst2];
par0=(inst'*inst)\inst'*y;

%% Iterative search using the Optimization Toolbox


% Options for the lsqcurvefit function
opt = optimset(...
'MaxIter',find(pl, 'MaxIter'),...
'TolX',find(pl, 'TolXm'),...
'TolFun',find(pl, 'TolFun'),...
'MaxFunEvals',find(pl, 'MaxFunEvals'),...
'Display','off');

% Upper and Lower Bounds for the parameters
ub=find(pl,'UpBound');
lb=find(pl,'LowBound');
if isempty(ub)
ub=ones(p+q,1);
pl = append(pl, param('UpBound', ub));
disp('! Parameters Upper Bound set to 1')
```

```
end
if isempty(lb)
lb=-ones(p+q,1);
pl = append(pl, param('LowBound', lb));
disp('! Parameters Lower Bound set to -1')
end

% Call to the Optimization Toolbox function
[par,res]=lsqcurvefit(@(par,xdata)filter([par(1:p)],[1 par(p+1:p+q)],xdata),...
       par0,x,y,lb,ub,opt);


%% Build output ao

% New output history
h = history(ALGONAME, VERSION, pl,[as(1).hist as(2).hist]);
h = set(h,'invars', invars);

% Make output analysis object
b = ao([par; res]);

% Name, mfilename, description for this object
b = setnh(b,...
   'name', ['ARMA param. Input: ' sprintf('%s ', char(invars{1})) ' Output: '
char(invars{2})],...
   'mfilename', ALGONAME, ...
   'description', find(pl,'description'),...
   'hist', h);

varargout{1} = b;


%% --- FUNCTIONS ---


% Get default parameters
function plo = getDefaultPL();
disp('* creating default plist...');
plo = plist();
plo = append(plo, param('MaxIter',4e2));
plo = append(plo, param('MaxFunEvals',1e2));
plo = append(plo, param('TolX',1e-6));
plo = append(plo, param('TolFun',1e-6));
plo = append(plo, param('ARMANum',2));
plo = append(plo, param('ARMADen',1));
disp('* done.');
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

pri2fq

PURPOSE ^

Convert comlpex pole/zero into frequency/Q pole/zero representation.

SYNOPSIS ^

function [f0, q]= pri2fq(c)

DESCRIPTION ^

```
Convert comlpex pole/zero into frequency/Q pole/zero representation.

 [f0, q]= pri2fq(c)

M Hewitson 2003-03-12


$Id: pri2fq.m,v 1.3 2008/01/17 16:00:06 anneke Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function [f0, q]= pri2fq(c)

% Convert comlpex pole/zero into frequency/Q pole/zero representation.
%
%  [f0, q]= pri2fq(c)
%
% M Hewitson 2003-03-12
%
%
% $Id: pri2fq.m,v 1.3 2008/01/17 16:00:06 anneke Exp $
```

```
%


if(nargin==0)
  disp('usage: [f0, q]= ri2fq(c)');
  return
end

a = real(c(1));
b = imag(c(1));

% f0 = -a*sqrt(b^2/a^2 + 1) / 2 / pi;
% q = sqrt(b^2/a^2 +1)/2;

k = -a*sqrt(b^2/a^2 + 1) * 2 * pi;
f0 = 1/k;
q = sqrt(b^2/a^2 +1)/2;

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

tomiir

PURPOSE ^

TOMIIR converts a pzmodel to an IIR filter using a bilinear transform.

SYNOPSIS ^

function f = tomiir(varargin)

DESCRIPTION ^

```
 TOMIIR converts a pzmodel to an IIR filter using a bilinear transform.

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

 DESCRIPTION: TOMIIR converts a pzmodel to an IIR filter using a bilinear
              transform.

 CALL:        f = tomiir(pzm, fs); % construct for this sample frequency fs
              f = tomiir(pzm, pl); % construct from plist

 PARAMETERS:  'fs'  - sample frequency

 VERSION:     $Id: tomiir.m,v 1.6 2008/02/12 20:07:37 hewitson Exp $

 HISTORY:     03-04-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pzmodel object into a string.

This function is called by:

SOURCE CODE ^

```
function f = tomiir(varargin)
% TOMIIR converts a pzmodel to an IIR filter using a bilinear transform.
%
```

```
%%%
```

```
    pl = varargin{2};
    fs = find(pl, 'fs');
    if isempty(fs)
      error('### unknown parameter list.');
    end
  else
    fs = varargin{2};
  end
end

% get a and b coefficients
[a,b] = pzm2ab(pzm, fs);

% make MIIR filter
f = miir(a,b,fs);


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_chkts

PURPOSE ^

LTPDA_CHKTS checks that the time-series in the input AO(s) are evenly sampled according to the sample rate and an allowed error.

SYNOPSIS ^

function varargout = ltpda_chkts(varargin)

DESCRIPTION ^

```
 LTPDA_CHKTS checks that the time-series in the input AO(s) are evenly sampled
according to the sample rate and an allowed error.

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

 DESCRIPTION: LTPDA_CHKTS checks that the time-series in the input AO(s) are
              evenly sampled according to the sample rate and an allowed error.

 CALL:        res = ltpda_chkts(a)
              res = 0  - time-stamps samples don't correspond to evenly sampled data
              res = 1  - time-stamps correspond to evenly sampled data

 GUI CALL:    The following call returns a parameter list object that contains
              the default parameter values:

              >> pl = ltpda_xcorr('Params')

 VERSION:     $Id: ltpda_chkts.m,v 1.4 2007/11/07 15:57:27 ingo Exp $

 HISTORY:     02-05-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = ltpda_chkts(varargin)
% LTPDA_CHKTS checks that the time-series in the input AO(s) are evenly sampled according to the sample rate and an allowed error.
%
%%%
```

```
na = length(as);

% check each
res = ones(na,1);
for j=1:na
  a  = as(j);
  d  = a.data;
  if isa(d, 'tsdata')
    fs    = d.fs;
    t     = d.x;
    tstep = 1/fs;
    % Allowed error is 1% of a sample step
    Terr  = 0.01*tstep;

    tt = 0;
    % go through time samples
    for k=1:length(t)
      tdiff = abs(t(k) - tt);
      if tdiff > Terr
        res(j) = 0;
        break
      end
      tt = tt + tstep;
    end
  else
    error('### I only work with time-series AOs at the moment.');
  end

end

varargout{1} = res;



%-----------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

disp('* creating default plist...');
plo = plist();
disp('* done.');

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

pri2fq

PURPOSE ^

Convert comlpex pole/zero into frequency/Q pole/zero representation.

SYNOPSIS ^

function [f0, q]= pri2fq(c)

DESCRIPTION ^

```
Convert comlpex pole/zero into frequency/Q pole/zero representation.

 [f0, q]= pri2fq(c)

M Hewitson 2003-03-12


$Id: pri2fq.m,v 1.3 2008/01/17 15:59:42 anneke Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function [f0, q]= pri2fq(c)

% Convert comlpex pole/zero into frequency/Q pole/zero representation.
%
%  [f0, q]= pri2fq(c)
%
% M Hewitson 2003-03-12
%
%
% $Id: pri2fq.m,v 1.3 2008/01/17 15:59:42 anneke Exp $
```

```
%


if(nargin==0)
  disp('usage: [f0, q]= ri2fq(c)');
  return
end

a = real(c(1));
b = imag(c(1));

k = -a*sqrt(b^2/a^2 + 1) * 2 * pi;
f0 = 1/k;
q = sqrt(b^2/a^2 +1)/2;



% f0 = -a*sqrt(b^2/a^2 + 1) / 2 / pi;
% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_classes

PURPOSE ^

LTPDA_CLASSES lists all the LTPDA object types.

SYNOPSIS ^

function classes = ltpda_classes(varargin)

DESCRIPTION ^

```
 LTPDA_CLASSES lists all the LTPDA object types.

%%%
```

```
% LTPDA_CLASSES lists all the LTPDA object types.
%
%%%
```

provenance

PURPOSE ^

PROVENANCE constructors for provenance class.

SYNOPSIS ^

function p = provenance(varargin)

DESCRIPTION ^

```
 PROVENANCE constructors for provenance class.

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

 DESCRIPTION:  PROVENANCE constructors for provenance class.

 CONSTRUCTORS: p = provenance();
               p = provenance('creator');

 VERSION:      $Id: provenance.m,v 1.19 2008/02/15 17:35:55 ingo Exp $

 HISTORY:      07-05-2007 M Hewitson
                  Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a provenance object into a string.
- provenance PROVENANCE constructors for provenance class.

This function is called by:

- provenance PROVENANCE constructors for provenance class.
- subsasgn SUBSASGN define index assignment for provenance properties.

SOURCE CODE ^

```
function p = provenance(varargin)
% PROVENANCE constructors for provenance class.
```

```
%
%%%
```

```
  %%%%%%%%%%% From File %%%%%%%%%%%%%%%%
  elseif ischar(varargin{1})

    filename = varargin{1};
    [path, name, ext, vers] = fileparts(filename);

    if ~isempty(ext)
      switch ext
        case '.mat'
          p = load(filename);
          p = p.a;
        case '.xml'
          root_node = xmlread(filename);
          p = ltpda_xmlread(root_node, 'provenance');
        otherwise
          error('### Unknown file type.');
      end
      return
    else
      % Set creator
      creator = varargin{1};
    end
  %%%%%%%%%% Create from structure %%%%%%%%%%%%%%
  elseif isstruct(varargin{1})

    p = class(varargin{1}, 'provenance');
    return

  %%%%%%%%%% Create from structure %%%%%%%%%%%%%%
  elseif isa(varargin{1}, 'plist')

    creator = find(varargin{1}, 'creator');
    creator = strcat('Recreate from: ', creator);
    p = provenance(creator);
    return
  else
    error('### incorrect inputs');
  end
end
created = time;

%-----------------------------------------------------------------------
% User Info
%
try
  if isempty(creator)
    creator = char(java.lang.System.getProperties.getProperty('user.name'));
  end
catch
  creator = 'unknown';
end

```

```
%-----------------------------------------------------------------------
% Machine Info
%
ipaddress = char(getHostAddress(java.net.InetAddress.getLocalHost));
hostname  = char(getHostName(java.net.InetAddress.getLocalHost));
os        = computer;

%-----------------------------------------------------------------------
% Process software versions
%

matlab_version  = getappdata(0, 'matlab_version');
sigproc_version = getappdata(0, 'sigproc_version');
ltpda_version   = getappdata(0, 'ltpda_version');

%-----------------------------------------------------------------------
% Make provenance object
%

p.creator         = creator;
p.created         = created;
p.ip              = ipaddress;
p.hostname        = hostname;
p.os              = os;
p.matlab_version  = matlab_version;
p.sigproc_version = sigproc_version;
p.ltpda_version   = ltpda_version;
p = class(p, 'provenance');

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

tsdata

PURPOSE ^

TSDATA time-series object class constructor.

SYNOPSIS ^

function ts = tsdata(varargin)

DESCRIPTION ^

```
 TSDATA time-series object class constructor.

%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

- reshapeT RESHAPET reshape the time-vector to match the x vector.
- setTime SETTIME sets the time-vector of a tsdata object based on the length of the x vector and the sample rate.

This function is called by:

- subsasgn SUBSASGN define index assignment for for tsdata properties.

SUBFUNCTIONS ^

- function ts = init

SOURCE CODE ^

```
function ts = tsdata(varargin)
% TSDATA time-series object class constructor.
%
%%%
```

```
%                            sample rate. The first sample is assigned time 0.
%
% HISTORY: 30-01-2007 Hewitson
%             Creation
%
%%%
```

```
    filename = varargin{1};
    [path, name, ext, vers] = fileparts(filename);
    switch ext
      case '.mat'
        ts = load(filename);
        ts = ts.a;
      case '.xml'
        root_node = xmlread(filename);
        ts = ltpda_xmlread(root_node, 'tsdata');
      otherwise
        error('### Unknown file type.');
    end
  %%%%%%%%%%  ts = tsdata(tsdata)   %%%%%%%%%%
  elseif isa(varargin{1}, 'tsdata')
    ts = varargin{1};

  %%%%%%%%%%  ts = tsdata(struct)   %%%%%%%%%%
  elseif isstruct(varargin{1})

    ts = init();

    fields = fieldnames(varargin{1});
    for ii = 1:length(fields)
      field = fields{ii};

      %%% created -> time-object
      if strcmp(field, 'created')
        created = varargin{1}.created;
        if isstruct(created)
          created = time(created);
        end
        ts.created     = created;
      %%% t0 -> time-object
      elseif strcmp(field, 't0')
        t0 = varargin{1}.t0;
        if isstruct(t0)
          t0 = time(t0);
        end
        ts.t0 = t0;
      %%% All other
      else
        try
          ts.(field) = varargin{1}.(field);
        catch
          error('### The field ''%s'' in the struct is not a tsdata property.', field)
        end
      end
    end

  %%%%%%%%%%  ts = tsdata(y_vector)   %%%%%%%%%%
  else

```

```
    ts = init();

    ts.y     = varargin{1};
    ts.fs    = 1;
    ts.nsecs = length(ts.y)/ts.fs;

    % Unify the y-axis
    if size(ts.y,2) > size(ts.y,1)
      ts.y = [ts.y].';
    end
    ts = setTime(ts);

  end

elseif nargin == 2

  %%%%%%%%%%% From DATABASE
  if isa(varargin{1}, 'database')
    ts = retrieve(varargin{1}, varargin{2:end});
    return
  %%%%%%%%%%  ts = tsdata(x_vector, y_vector)   %%%%%%%%%%
  elseif length(varargin{1}) == length(varargin{2}) && length(varargin{1})>1

    ts = init();

    ts.x       = varargin{1};
    ts.y       = varargin{2};
    ts.fs      = 1.0/(ts.x(min(2,length(ts.x)))-ts.x(1));
    ts.nsecs   = ts.x(end) - ts.x(1) + 1/ts.fs;

    % Unify the y-axis
    if size(ts.y,2) > size(ts.y,1)
      ts.y = [ts.y].';
    end
    ts = reshapeT(ts);

  %%%%%%%%%%  ts = tsdata(y_vector, fs)   %%%%%%%%%%
  else
    % check we have fs properly here
    if length(varargin{2}) > 1
      error(['### unknown constructor call. Either x and y should be the' ...
        'same length, or fs should be a single value.']);
    end

    ts = init();

    ts.y       = varargin{1};
    ts.fs      = varargin{2};
    ts.nsecs   = length(ts.y)/ts.fs;

    if size(ts.y,2) > size(ts.y,1)
      ts.y = [ts.y].';
```

```
    end
    ts = setTime(ts);

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


end % function ts = tsdata(varargin)
```

Generated on Tue 26-Feb-2008 10:52:52 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 a1 and all other
              input AOs. Therefore, if the input argument list contains N
              analysis objects, the output, b, will contain N-1 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]
              Nolap - segment overlap (number of samples) [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 1.10 2007/12/18 14:11:26 josep Exp $
```

```
 HISTORY:    07-02-2007 M Hewitson
                Creation

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = ltpda_cohere('Params')

%%%
```

```
%               Nolap - segment overlap (number of samples) [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 1.10 2007/12/18 14:11:26 josep Exp $
%
% HISTORY:    07-02-2007 M Hewitson
%                Creation
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = ltpda_cohere('Params')
%
%%%
```

```
  pl = getDefaultPL();
else
  pl = combine(ps, getDefaultPL);
end

varargout{1} = ltpda_xspec(as, pl, 'COHERE', ALGONAME, VERSION, invars);


%-----------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

disp('* creating default plist...');
plo = plist();
plo = append(plo, param('Nfft', -0.5));
plo = append(plo, param('Win', specwin('Kaiser', 1, 100)));
plo = append(plo, param('Nolap', -1));
plo = append(plo, param('Order', 0));

disp('* done.');
```

Generated on Tue 26-Feb-2008 10:52:52 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 ^

```
0001
0002 function hh = uitabpanel(varargin)
0003
0004 % UITABPANEL adds a group of tabbed uipanel container objects to a figure.
0005 %   The user can click on a tab with the left mouse button to select it.
0006 %   One tab is always selected, and appears on top of all the orthers.
0007 %   The syntax is the same as UIPANEL except the following property changes:
0008 %
0009 %     Property              Values
0010 %     -------------------   ------------------------------------------------
0011 %     Title                 a cell array of strings specifying the title of
0012 %                           each tab
0013 %     String                (same as Title)
0014 %     TitlePosition         the location of tabs in relation of the visible
0015 %                           panel, the value is the same as UIPANEL
0016 %     Style                 (same as TitlePosition)
0017 %     FrameBackgroundColor  the background color for the tabpanel frame
0018 %     FrameBorderType       the border type for the tabpanel frame
```

```
%     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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_compute_f

PURPOSE ^

LTPDA_COMPUTE_F compute the frequency and resolution vectors for use with mlpsd.

SYNOPSIS ^

function [f,r,m, L,rr,rrr] = ltpda_compute_f(fs, N, Kdes, Kmin, Jdes, olap, varargin)

DESCRIPTION ^

```
 LTPDA_COMPUTE_F compute the frequency and resolution vectors for use with mlpsd.

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

 DESCRIPTION: LTPDA_COMPUTE_F compute the frequency and resolution vectors for
              use with mlpsd.

 CALL:        [f,r,m,L] = ltpda_compute_f(fs, N, Kdes, Kmin, Jdes, olap)

 INPUTS:      fs   - sample frequency of data
              N    - length of data vector
              Kdes - desired number of averages
              Kmin - minimun number of averages
              Jdes - desired number of points in spectrum
              olap - overlap value (0-1)

 VERSION:    $Id: ltpda_compute_f.m,v 1.4 2007/10/28 10:23:40 hewitson Exp $

 HISTORY:    22-01-2007 M Hewitson
                Creation

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = ltpda_compute_f('Params')

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpda_lpsd LTPDA_LPSD implement LPSD algorithm for analysis objects.
- ltpda_ltfe LTPDA_LTFE implement transfer-function estimation computed on a log frequency axis.

SUBFUNCTIONS ^

- function pl = getDefaultPL()

SOURCE CODE ^

```
function [f,r,m, L,rr,rrr] = ltpda_compute_f(fs, N, Kdes, Kmin, Jdes, olap, varargin)
% LTPDA_COMPUTE_F compute the frequency and resolution vectors for use with mlpsd.
%
%%%
```

```
  %%%%%%%%%%%% Do linear frequency for testing

  f   = [linspace(0, fs/2, Jdes/2 + 1)].';
  L   = Jdes * ones(size(f));
  r   = fs./L;
  m   = f./r;
  r   = [];
  rr  = [];
  rrr = [];


else
  %%%%%%%%%%%% Do log frequency

  % Settings
  fmin = fs/N;
  fmax = fs/2;
  j    = 1;
  f(j) = fmin;

  % Initialise constants
  rav  = fs/N*(1+(1-olap)*(Kdes-1));
  rmin = fs/N*(1+(1-olap)*(Kmin-1));
  g    = log(fmax)-log(fmin);

  % compute f(j)
  while f < fmax
    % compute r'(j)
    rr(j) = f(j)*g/(Jdes-1);
    % compute r''(j)
    if rr(j) >= rav
      rrr(j) = rr(j);
    else
      rrr(j) = sqrt(rav*rr(j));
    end

    % if this is below the lower possible resolution
    if rrr(j) <= rmin
      rrr(j) = rmin;
    end

    % compute L(j)
    L(j) = round(fs/rrr(j));

    % compute freq res
    r(j) = fs/L(j);
    m(j) = f(j)/r(j);

    tf = f(j) + r(j);
    if tf<fmax
      f(j+1) = tf;
      j = j + 1;
```

```
    else
      break
    end
  end
end


%-----------------------------------------------------------------------
% get default parameter list
function pl = getDefaultPL()

pl = plist();
pl = append(pl, param('fs',   []));
pl = append(pl, param('N',    []));
pl = append(pl, param('Kdes', []));
pl = append(pl, param('Kmin', []));
pl = append(pl, param('Jdes', []));
pl = append(pl, param('olap', []));


```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

pzm2ab

PURPOSE ^

PZM2AB convert pzmodel to IIR filter coefficients using bilinear

SYNOPSIS ^

function [ao,bo] = pzm2ab(pzm, fs)

DESCRIPTION ^

```
PZM2AB convert pzmodel to IIR filter coefficients using bilinear
transform.

usage: [a,b] = pzm2ab(pzm, fs)

M Hewitson 03-04-07

$Id: pzm2ab.m,v 1.4 2007/09/03 09:27:04 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

- abcascade Cascade two filters together to get a new filter.

This function is called by:

SUBFUNCTIONS ^

- function [a,b] = cpolezero(pf, pq, zf, zq, fs)
- function g = iirdcgain(a,b)

SOURCE CODE ^

```
function [ao,bo] = pzm2ab(pzm, fs)
% PZM2AB convert pzmodel to IIR filter coefficients using bilinear
% transform.
%
% usage: [a,b] = pzm2ab(pzm, fs)
%
% M Hewitson 03-04-07
```

```
%
% $Id: pzm2ab.m,v 1.4 2007/09/03 09:27:04 hewitson Exp $
%

disp(sprintf('$$$ converting %s', get(pzm, 'name')))

gain  = get(pzm, 'gain');
poles = get(pzm, 'poles');
zeros = get(pzm, 'zeros');
np = length(poles);
nz = length(zeros);

ao = [];
bo = [];

% First we should do complex pole/zero pairs
cpoles = [];
for j=1:np
  if get(poles(j), 'q') > 0.5
    cpoles = [cpoles poles(j)];
  end
end
czeros = [];
for j=1:nz
  if get(zeros(j), 'q') > 0.5
    czeros = [czeros zeros(j)];
  end
end

czi = 1;
for j=1:length(cpoles)
  if czi <= length(czeros)
    % we have a pair
    p = cpoles(j);
    z = czeros(czi);

    [ai,bi] = cpolezero(get(p, 'f'), get(p, 'q'), get(z, 'f'), get(z, 'q'), fs);
    if ~isempty(ao)>0
      [ao,bo] = abcascade(ao,bo,ai,bi);
    else
      ao = ai;
      bo = bi;
    end

    % increment zero counter
    czi = czi + 1;
  end
end

if length(cpoles) > length(czeros)
  % do remaining cpoles
  for j=length(czeros)+1:length(cpoles)
    disp('$$$ computing complex pole');
```

```
    [ai,bi] = cp2iir(cpoles(j), fs);
    if ~isempty(ao)>0
      [ao,bo] = abcascade(ao,bo,ai,bi);
    else
      ao = ai;
      bo = bi;
    end
  end
else
  % do remaining czeros
  for j=length(cpoles)+1:length(czeros)
    disp('$$$ computing complex zero');
    [ai,bi] = cz2iir(czeros(j), fs);
    if ~isempty(ao)>0
      [ao,bo] = 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(get(pole, 'q')) || get(pole, 'q') < 0.5
    disp('$$$ computing real pole');
    [ai,bi] = rp2iir(pole, fs);
    if ~isempty(ao)>0
      [ao,bo] = abcascade(ao,bo,ai,bi);
    else
      ao = ai;
      bo = bi;
    end
  end
end

for j=1:nz
  zero = zeros(j);
  if isnan(get(zero, 'q')) || get(zero, 'q') < 0.5
    disp('$$$ computing real zero');
    [ai,bi] = rz2iir(zero, fs);
    if ~isempty(ao)>0
      [ao,bo] = abcascade(ao,bo,ai,bi);
    else
      ao = ai;
      bo = bi;
    end
  end
end

% Old version that did everything individually - not so good.
```

```
% for j=1:np
%   pole = poles(j);
%   if get(pole, 'q') > 0.5
%     [ai,bi] = cp2iir(pole, fs);
%   else
%     [ai,bi] = rp2iir(pole, fs);
%   end
%   if ~isempty(ao)>0
%     [ao,bo] = abcascade(ao,bo,ai,bi);
%   else
%     ao = ai;
%     bo = bi;
%   end
% end
%
% for j=1:nz
%   zero = zeros(j);
%   if get(zeros, 'q') > 0.5
%     [ai,bi] = cz2iir(zero, fs);
%   else
%     [ai,bi] = rz2iir(zero, fs);
%   end
%   if ~isempty(ao)>0
%     [ao,bo] = abcascade(ao,bo,ai,bi);
%   else
%     ao = ai;
%     bo = bi;
%   end
% end

ao = ao.*gain;

function [a,b] = cpolezero(pf, pq, zf, zq, fs)
%
% Return IIR filter coefficients for a complex pole
% and complex zero designed using the bilinear transform.
%
% usage: [a,b] = cpolezero(pf, pq, zf, zq, fs)
%
%
%  M Hewitson 2003-02-18
%

disp('$$$ computing complex pole/zero pair');

wp = pf*2*pi;
wp2 = wp^2;
wz = zf*2*pi;
wz2 = wz^2;

k = 4*fs*fs + 2*wp*fs/pq + wp2;

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;


function g = iirdcgain(a,b)

% Work out the DC gain of an IIR
% filter given the coefficients.
%
% usage:
%   g = iirdcgain(a,b)
%
% inputs:
%   a - numerator coefficients
%   b - denominator coefficients
%
% outputs:
%   g - gain
%
% M Hewitson 03-07-02
%
%
% $Id: pzm2ab.m,v 1.4 2007/09/03 09:27:04 hewitson Exp $
%

g = 0;

suma = sum(a);
if(length(b)>1)
  sumb = sum(b);
  g = suma / sumb;
else
  g = suma;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY implement terminal display for cdata object.

SYNOPSIS ^

function txt = display(cdatas, varargin)

DESCRIPTION ^

```
 DISPLAY implement terminal display for cdata object.

%%%
```

```
%
% VERSION: $Id: display.m,v 1.7 2008/02/12 19:54:05 hewitson Exp $
%
% HISTORY: 31-01-2007 M Hewitson
%             Creation
%
%%%
```

```
  end

  txt{end+1} = ['  x: ' tstr];

  banner_end(1:length(banner)) = '-';
  txt{end+1} = banner_end;

  txt{end+1} = ' ';
  txt{end+1} = ' ';
end

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 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 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 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]
             'Nolap' - segment overlap (number of samples) [default: taken from
window function]
             'Nfft'  - number of samples in each fft [default: fs of input data]
             'Debug' - debug level for terminal output (0-1)

 VERSION:     $Id: ltpda_cpsd.m,v 1.7 2007/07/16 12:52:21 ingo Exp $

 HISTORY:     07-02-2007 M Hewitson
                 Creation

 The following call returns a parameter list object that contains the
 default parameter values:
```

```
>> pl = ltpda_cpsd('Params')

%%%
```

```
%
% VERSION:     $Id: ltpda_cpsd.m,v 1.7 2007/07/16 12:52:21 ingo Exp $
%
% HISTORY:     07-02-2007 M Hewitson
%                 Creation
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = ltpda_cpsd('Params')
%
%%%
```

```
% Get default params
function plo = getDefaultPL()

disp('* creating default plist...');
plo = plist();
plo = append(plo, param('Nfft', -0.5));
plo = append(plo, param('Win', specwin('Kaiser', 1, 100)));
plo = append(plo, param('Nolap', -1));
disp('* done.');


```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

utp_ltpda_simulate_discrete

PURPOSE ^

tests ltpda_simulate_discrete which simulates a discrete state space system

SYNOPSIS ^

function result = utp_ltpda_kalman_filter

DESCRIPTION ^

```
tests ltpda_simulate_discrete which simulates a discrete state space system
%%%
```

```
function result = utp_ltpda_kalman_filter
%tests ltpda_simulate_discrete which simulates a discrete state space system
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: tests ltpda_simulate_discrete whichsimulates a discrete
% state space system
%
% CALL: utp_ltpda_simulate_discrete
%
% INPUTS:
%
% OUTPUTS: result - 1/0
%
% ***** THERE ARE NO DEFAULT PARAMETERS *****
%
% &&VERSION: $Id: utp_ltpda_simulate_discrete.m,v 1.1 2008/02/20 09:20:07 adrien Exp $
%
% HISTORY: 17-02-2008 A Grynagier
%
% TO DO: Numerical and syntaxic tests
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ALGONAME = mfilename;
VERSION = '$Id: utp_ltpda_simulate_discrete.m,v 1.1 2008/02/20 09:20:07 adrien Exp $';
result = 1;
display(['starting ' mfilename]);

%% create subsystem
%% test numerique
A=expm([0 1; -2 -1]);
B1=[0;1];
C=[2 -1];
D1=1;
B2=[0;0];
D2=1;
sigma = 0.5;
mean = 10;
syst = plist('TYPE', 'SUBSYSTEM' ,'NAME', 'SS1' ,'TIMESTEP', 1 , ...
    'XISOUTPUT',0,'YISOUTPUT',1,'XINI', [0 0]' , ...
    'PARAMNAMES', {} ,'PARAMVALUES', [] ,'PARAMSIGMAS', [] ,...
    'NBINPUTS', 2 ,'INPUTNAMES', {'U' 'N'} ,'INPUTSIZES', [1 1] , 'INPUTISUSED', [1 1] ,...
    'AMAT', {A} ,'BMATS', {B1 B2} ,'CMAT', {C} ,'DMATS', {D1 D2} );
ao1 = ao(sin((1:100)/3));
ao_in_list = plist('U',ao1, 'N', plist('SIGMA', sigma, 'MEAN',mean));
ao_out = ltpda_simulate_discrete(ao_in_list, syst);
%
data_plist = plist('Qs',{0},'Rs',{sigma^2+mean^2},'Noise_Names',{'N'} );

ao_in_list2 = plist('U',ao1);
```

```
ao_id = ltpda_ss_kalman_filter(ao_out, ao_in_list2, data_plist, syst);

figure(1)
plot(ao_out)
figure(2)
plot(ao_id)
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY implement terminal display for fsdata object.

SYNOPSIS ^

function txt = display(fsdatas, varargin)

DESCRIPTION ^

```
 DISPLAY implement terminal display for fsdata object.

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

 DESCRIPTION: DISPLAY implement terminal display for fsdata object.

 VERSION: $Id: display.m,v 1.8 2008/02/12 22:11:21 hewitson Exp $

 HISTORY: 31-01-2007 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- get GET get a fsdata property.

This function is called by:

- set SET set a fsdata property.

SOURCE CODE ^

```
function txt = display(fsdatas, varargin)
% DISPLAY implement terminal display for fsdata object.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: DISPLAY implement terminal display for fsdata object.
```

```
%
% VERSION: $Id: display.m,v 1.8 2008/02/12 22:11:21 hewitson Exp $
%
% HISTORY: 31-01-2007 M Hewitson
%             Creation
%
%%%
```

```
  banner_end(1:length(banner_start)) = '-';
  txt{end+1} = banner_end;

  txt{end+1} = ' ';
  txt{end+1} = ' ';
end

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_cscale

PURPOSE ^

LTPDA_CSCALE Set the color range of the current figure

SYNOPSIS ^

function ltpda_cscale(y1,y2)

DESCRIPTION ^

```
 LTPDA_CSCALE Set the color range of the current figure

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

 DESCRIPTION: LTPDA_CSCALE Set the color range of the current figure

 CALL:        cscale(x1,x2)

 VERSION:     $Id: ltpda_cscale.m,v 1.4 2007/07/16 12:52:20 ingo Exp $

 HISTORY:     09-10-2001 M Hewitson
                 Creation.

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_cscale(y1,y2)
% LTPDA_CSCALE Set the color range of the current figure
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DESCRIPTION: LTPDA_CSCALE Set the color range of the current figure
%
% CALL:        cscale(x1,x2)
%
```

```
% VERSION:     $Id: ltpda_cscale.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
%
% HISTORY:     09-10-2001 M Hewitson
%                 Creation.
%
%%%
```

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 1.8 2008/01/22 20:45:23 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 1.8 2008/01/22 20:45:23 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 = pole(f,q);
else
  p = pole(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 = zero(f,q);
else
  z = zero(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 = get(p, 'f');
  q = get(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 = get(p, 'f');
  q = get(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

% 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.xilterStrEdit, 'String', string(pzm));

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, ltpda_phase(my), fx, ltpda_phase(fy), 'r--')
grid on;
axis tight
xlabel('Frequency [Hz]');
```

```
ylabel('Phase [deg]');



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);
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


filtresp = resp(filt, plist(param('f', mx.')));
d        = filtresp.data;
fx       = d.x;
fy       = d.y;

figure
subplot(3,1,1:2)
loglog(mx,abs(my), fx, abs(fy), 'r--')
grid on;
axis tight
xlabel('');
ylabel('Magnitude');
legend('Pole/Zero model', 'IIR','Location','SouthWest')

subplot(3,1,3)
semilogx(mx, ltpda_phase(my), fx, ltpda_phase(fy), 'r--')
grid on;
axis tight
xlabel('Frequency [Hz]');
ylabel('Phase [deg]');

```

```

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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

utp_ltpda_ss_assemble_1

PURPOSE ^

test function for utp_ltpda_ss_assemble using 3 subsystems, random dims.

SYNOPSIS ^

function output = utp_ltpda_ss_assemble_1

DESCRIPTION ^

```
 test function for utp_ltpda_ss_assemble using 3 subsystems, random dims.
%%%
```

SOURCE CODE ^

```
function output = utp_ltpda_ss_assemble_1
% test function for utp_ltpda_ss_assemble using 3 subsystems, random dims.
%%%
```

```
    B2 = rand(ss2,in2);
    C2 = rand(out2,ss2);
    D2 = rand(out2,in2);
    A3 = rand(ss3,ss3);
    B3 = rand(ss3,out1);
    C3 = rand(out3,ss3);
    D3 = rand(out3,out1);
    E3 = rand(ss3,out2);
    F3 = rand(out3,out2);
    o3 = eye(out3);
    A_meta = {A1,[],[];[],A2,[];[],[],A3};
    B_meta = {[],[],[];[],[],[];B3,E3,[]};
    C_meta = {C1,[],[];[],C2,[];[],[],C3};
    D_meta = {[],[],[];[],[],[];D3,F3,[]};
    Bext_meta = {B1,[];[],B2;[],[]};
    Dext_meta = {D1,[];[],D2;[],[]};
    Ox_meta = {[],[],[]};
    Oy_meta = {[],[],o3};

%% local subsystems and call
    subsyst1 = plist('TYPE', 'SUBSYSTEM' ,'NAME', 'Y1' ,'TIMESTEP', 0 , ...
    'XISOUTPUT',0,'YISOUTPUT',0,'XINI', zeros(ss1,1) , ...
    'PARAMNAMES', {'p1' 'p2' 'p3'} ,'PARAMVALUES', [1 2 3] ,'PARAMSIGMAS', [2 5 3] ,...
    'NBINPUTS', 1 ,'INPUTNAMES', {'I1'} ,'INPUTSIZES', in1 , 'INPUTISUSED', 1 ,...
    'AMAT', {A1} ,'BMATS', {B1} ,'CMAT', {C1} ,'DMATS', {D1} );
    subsyst2 = plist('TYPE', 'SUBSYSTEM' ,'NAME', 'Y2' ,'TIMESTEP', 0 , ...
    'XISOUTPUT',0,'YISOUTPUT',0,'XINI', zeros(ss2,1) , ...
    'PARAMNAMES', {'p1'} ,'PARAMVALUES', [1] ,'PARAMSIGMAS', [2] ,...
    'NBINPUTS', 1 ,'INPUTNAMES', {'I2'} ,'INPUTSIZES', in2 , 'INPUTISUSED', 1 ,...
    'AMAT', {A2} ,'BMATS', {B2} ,'CMAT', {C2} ,'DMATS', {D2} );
    subsyst3 = plist('TYPE', 'SUBSYSTEM' ,'NAME', 'Y3' ,'TIMESTEP', 0 , ...
    'XISOUTPUT',0,'YISOUTPUT',1,'XINI', zeros(ss3,1) , ...
    'PARAMNAMES', {'p4' 'p2' 'p1'} ,'PARAMVALUES', [4 2 1] ,'PARAMSIGMAS', [4 5 2] ,...
    'NBINPUTS', 2 ,'INPUTNAMES', {'Y1'; 'Y2'} ,'INPUTSIZES', [out1 out2] , 'INPUTISUSED', [1 1] ,...
    'AMAT', {A3} ,'BMATS', {B3; E3} ,'CMAT', {C3} ,'DMATS', {D3; F3} );
    ltpda_ss_check(subsyst1);
    ltpda_ss_check(subsyst2);
    ltpda_ss_check(subsyst3);

%% meta system and fusionned matrices automatic generation
    listSubsystems = plist('Y1',subsyst1,'Y2',subsyst2,'Y3',subsyst3);
    listMeta = ltpda_ss_assemble_arrange(listSubsystems);
    [Subsys] = ltpda_ss_assemble_fusion(listMeta);
    A_mat = find(Subsys, 'AMAT');
    B_mats = find(Subsys, 'BMATS');
    C_mat = find(Subsys, 'CMAT');
    D_mats = find(Subsys, 'DMATS');
```

```

%% meta system and fusionned matrices by hand
    Xini_meta = {zeros(ss1,1) zeros(ss2,1) zeros(ss3,1)};
    SubsystemNames = {'Y1' 'Y2' 'Y3'};
    ParamNames_meta = {{'p1' 'p2' 'p3'} {'p1'} {'p4' 'p2' 'p1'}};
    ParamValues_meta = {[1 2 3] [1] [4 2 1]};
    ParamSigmas_meta = {[2 5 3] [2] [4 5 2]};
    NbInputs_meta = 2;
    InputNames_meta = {'I1' ;'I2'};
    InputSizes_meta = [in1 in2] ;
    IsUsed_meta =  [1 1];

    A_mat0 = [A1              zeros(ss1,ss2)  zeros(ss1,ss3); ...
        zeros(ss2,ss1)  A2              zeros(ss2,ss3); ...
        zeros(ss3,ss1)  zeros(ss3,ss2)  A3            ];
    B_mat0 = [zeros(ss1,out1)  zeros(ss1,out2)  zeros(ss1,out3); ...
        zeros(ss2,out1)  zeros(ss2,out2)  zeros(ss2,out3); ...
        B3               E3               zeros(ss3,out3)  ];
    C_mat0 = [C1               zeros(out1,ss2)  zeros(out1,ss3); ...
        zeros(out2,ss1)  C2               zeros(out2,ss3); ...
        zeros(out3,ss1)  zeros(out3,ss2)  C3          ];
    D_mat0 = [zeros(out1,out1) zeros(out1,out2) zeros(out1,out3); ...
        zeros(out2,out1) zeros(out2,out2) zeros(out2,out3); ...
        D3               F3               zeros(out3,out3)  ];
    Bext_mats0{1} = [ B1               ; zeros(ss2,in1)  ; zeros(ss3,in1)   ];
    Bext_mats0{2} = [ zeros(ss1,in2)   ; B2              ; zeros(ss3,in2)   ];
    Dext_mats0{1} = [D1                ; zeros(out2,in1) ; zeros(out3,in1)  ];
    Dext_mats0{2} = [ zeros(out1,in2)  ; D2              ; zeros(out3,in2)  ];
    Ox_mat0 = [zeros(out3,ss1) zeros(out3,ss2) zeros(out3,ss3)];
    Oy_mat0 = [zeros(out3,out1) zeros(out3,out2) o3];
    %take minimal system
    IDinv = inv(eye(size(D_mat0,1)) - D_mat0);
    A0 = A_mat0 + B_mat0*IDinv*C_mat0;
    C0 = Ox_mat0 + Oy_mat0*IDinv*C_mat0;
    BMATS0 = cell(2,1);
    DMATS0 = cell(2,1);
    for j_meta = 1:2
        BMATS0{j_meta} = Bext_mats0{j_meta} + B_mat0*IDinv*Dext_mats0{j_meta};
        DMATS0{j_meta} = Oy_mat0*IDinv*Dext_mats0{j_meta};
    end
    %retriveing fields out of plist
    % parsing the meta matrices into a list
    listMeta0 = plist('A_META', A_meta, 'BEXT_META', Bext_meta, 'C_META',
C_meta, 'DEXT_META', Dext_meta, 'D_META', D_meta, 'B_META', B_meta, ...
        'OY_META', Oy_meta, 'OX_META', Ox_meta,...
        'TIMESTEP', 0, 'XINI_META', Xini_meta, 'NAMES_META', SubsystemNames,...
        'PARAMNAMES_META', ParamNames_meta ,'PARAMVALUES_META',
ParamValues_meta ,'PARAMSIGMAS_META', ParamSigmas_meta ,...
        'NBINPUTS_META', NbInputs_meta ,'INPUTNAMES_META',
InputNames_meta ,'INPUTSIZES_META', InputSizes_meta , 'INPUTISUSED_META',
IsUsed_meta);

%% comparison between two results
```

```
    sumNorms =  norm(B_mats{1} - BMATS0{1}) + norm(B_mats{2} - BMATS0{2})  + norm(D_mats{1} - DMATS0{1})  + norm(D_mats{2} - DMATS0{2}) ...
        + norm(A_mat{1} - A0) + norm(C_mat{1} - C0);

    if not( isequal(0, sumNorms))
        display('erreur')
        output = 0;
    else
        display('OK')

    end
    display(count)
end
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY implement terminal display for history object.

SYNOPSIS ^

function txt = display(hists, varargin)

DESCRIPTION ^

```
 DISPLAY implement terminal display for history object.

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

 DESCRIPTION: DISPLAY implement terminal display for history object.

 CALL:        txt = display(history)

 INPUT:       history - history object

 OUTPUT:      txt     - cell array with strings to display the history object

 VERSION:     $Id: display.m,v 1.10 2008/02/12 22:30:27 hewitson Exp $

 HISTORY:     30-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- display DISPLAY implement terminal display for history object.
- get GET get history object properties.

This function is called by:

- display DISPLAY implement terminal display for history object.
- set SET set a history property.

SUBFUNCTIONS ^

- function new_txt = single_cell(txt_field)

SOURCE CODE ^

```
function txt = display(hists, varargin)
% DISPLAY implement terminal display for history object.
%
%%%
```

```
  banner = sprintf('-------- %s / %s / %d ------------', get(h, 'name'),    ...
                                                         get(h, 'version'), ...
                                                         get(h, 'n'));
  txt{end+1} = banner;

  txt{end+1} = ' ';
  txt{end+1} = sprintf('created: %s', char(h.created));

  % show input vars
  txt{end+1} = ['Inputs: ' get(h, 'invars')];

  % show parameter list
  txt{end+1} = 'Parameters';
  txt{end+1} = '----------';
  pl = get(h, 'plist');
  if ~isempty(pl)
    txt{end+1} = display(pl);
  end

  txt{end+1} = ' ';

  txt{end+1} = 'Histories';
  txt{end+1} = '---------';

  % show input histories
  inhists = get(h, 'inhists');
  nhists = length(inhists);
  txt{end+1} = sprintf('n input histories: %d', nhists);
  if nhists > 0
    txt{end+1} = display(inhists);
  end

  txt{end+1} = ' ';

  bannerEnd = [];
  while length(bannerEnd) < length(banner)
    bannerEnd = [bannerEnd '-'];
  end
  txt{end+1} = bannerEnd;

  txt{end+1} = ' ';
  txt{end+1} = ' ';
end

txt = single_cell(txt);

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end
```

```
0096
0097 %%%%
```

ltpda_dbquery

PURPOSE ^

LTPDA_DBQUERY query an AO repository database.

SYNOPSIS ^

function varargout = ltpda_dbquery(varargin)

DESCRIPTION ^

```
 LTPDA_DBQUERY query an AO repository database.

%%%
```

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

   The 'info' structure will contain a list of records resulting from the SQL query.

 VERSION:     $Id: ltpda_dbquery.m,v 1.5 2008/02/24 13:12:46 hewitson Exp $

 HISTORY:     10-05-2007 M Hewitson
                 Creation

%%%
```

```
%
% EXAMPLES:
%
%   >> 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');
%
%   >> info  = ltpda_dbquery(conn)
%
%   The 'info' structure will contain a list of the tables in the database.
%
%   >> info  = ltpda_dbquery(conn, query)
%
%   The 'info' structure will contain a list of records resulting from the SQL query.
%
% VERSION:     $Id: ltpda_dbquery.m,v 1.5 2008/02/24 13:12:46 hewitson Exp $
%
% HISTORY:     10-05-2007 M Hewitson
%                 Creation
%
%%%
```

```
end

% get table list
if nargin == 1
    info = getTableList(conn);
% execute query
elseif nargin == 2
  info = simpleQuery(conn, varargin{2});
% query a table
elseif nargin == 3

  table    = varargin{2};
  query    = varargin{3};
  info = runQuery(conn, table, query);

else
  error('### Incorrect inputs.');
end

varargout{1} = info;

%-----------------------------------------------------------------------
% Get table list
function info = getTableList(conn)

% open a connection
try
  curs = exec(conn, 'show tables');
  curs = fetch(curs);
  info = curs.Data;
  close(curs);
catch
  error('### Failed to get table list. Server returned: %s', curs.Message);
end

%-----------------------------------------------------------------------
% Get field list
function info = getFieldList(conn, table)

try
  curs = exec(conn, sprintf('describe %s', table));
  curs = fetch(curs);
  info = curs.Data;
  close(curs);
catch
  error('### Failed to get field list. Server returned: %s', curs.Message);
end

%-----------------------------------------------------------------------
% Get field list
function info = simpleQuery(conn, q)

```

```
% open a connection

try
  curs = exec(conn, sprintf('%s', q));
  curs = fetch(curs);
  info = curs.Data;
  close(curs);
catch
  error('### Failed to execute query. Server returned: %s', curs.Message);
end



%-----------------------------------------------------------------------
% Run a query
function info = runQuery(conn, table, query)

% Run query
info = [];

fieldlist = getFieldList(conn, table);
fields  = fieldlist(:,1);
f = '';
fs = {};
for j=1:length(fields)
  % special cases
  f=[f fields{j} ',' ];
  fs = [fs fields(j)];
end
q = sprintf('select %s  from %s where %s', f(1:end-1), table, query);
disp(['** QUERY: ' q]);
try
  curs = exec(conn, q);
  curs = fetch(curs);
  info = curs.Data;
  close(curs);
catch
  error('### Failed to query table. Server returned: %s', curs.Message);
end


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

utp_ltpda_ss_assemble_2

PURPOSE ^

test function for ltpda_ss_assemble - syntaxic

SYNOPSIS ^

function output = utp_ltpda_ss_assemble_2

DESCRIPTION ^

```
 test function for ltpda_ss_assemble - syntaxic
%%%
```

```
function output = utp_ltpda_ss_assemble_2
% test function for ltpda_ss_assemble - syntaxic
%%%
```

```
    catch
        display('error')
        output = 0;

    end
end
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY overloads display functionality for mfir objects.

SYNOPSIS ^

function txt = display(fs, varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for mfir objects.

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

 DESCRIPTION: DISPLAY overloads display functionality for mfir objects.

 CALL:        txt  = display(mfir)

 INPUT:       mfir - FIR filter object

 OUTPUT:      txt  - cell array with strings to display the mfir object

 VERSION:     $Id: display.m,v 1.8 2008/02/12 22:39:49 hewitson Exp $

 HISTORY:     11-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a mfir object into a string.
- display DISPLAY overloads display functionality for mfir objects.
- get GET get mfir properties.

This function is called by:

- display DISPLAY overloads display functionality for mfir objects.
- set SET set a mfir object property.

SOURCE CODE ^

```
function txt = display(fs, varargin)
% DISPLAY overloads display functionality for mfir objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: DISPLAY overloads display functionality for mfir objects.
%
% CALL:        txt  = display(mfir)
%
% INPUT:       mfir - FIR filter object
%
% OUTPUT:      txt  - cell array with strings to display the mfir object
%
% VERSION:     $Id: display.m,v 1.8 2008/02/12 22:39:49 hewitson Exp $
%
% HISTORY:     11-01-2007 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: display.m,v 1.8 2008/02/12 22:39:49 hewitson Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(fs, 'mfir') && ischar(varargin{1})
    in = char(varargin{1});
    if strcmp(in, 'Params')
      txt = plist();
      return
    elseif strcmp(in, 'Version')
      txt = VERSION;
      return
    elseif strcmp(in, 'Category')
      txt = CATEGORY;
      return
    end
  end
end

txt = {};

for i=1:numel(fs)
  f = fs(i);
  banner_start = sprintf('-------- %s / %s ------------', ...
                         get(f, 'name'),                  ...
                         get(f, 'version'));
  txt{end+1} = banner_start;

```

```
  txt{end+1} = ' ';
  txt{end+1} = sprintf('created: %s', char(f.created));
  txt{end+1} = sprintf('version: %s', f.version);
  txt{end+1} = sprintf('   name: %s', f.name);
  txt{end+1} = ['     fs: ' num2str(f.fs)];
  txt{end+1} = ['  ntaps: ' num2str(f.ntaps)];
  txt{end+1} = ['      a: ' num2str(f.a)];
  txt{end+1} = ['   gain: ' num2str(f.gain)];
  txt{end+1} = ['histout: ' num2str(f.histout)];

  txt{end+1} = sprintf('Parameters:');
  txt_plist = display(f.plist);

  for ii = 1:length(txt_plist)
    txt{end+1} = txt_plist{ii};
  end

  txt{end+1} = ' ';

  banner_end(1:length(banner_start)) = '-';
  txt{end+1} = banner_end;

  txt{end+1} = ' ';
  txt{end+1} = ' ';
end

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_deg2rad

PURPOSE ^

LTPDA_DEG2RAD Convert degrees to radians

SYNOPSIS ^

function r = ltpda_deg2rad(deg, min, sec)

DESCRIPTION ^

```
 LTPDA_DEG2RAD Convert degrees to radians

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

 DESCRIPTION: LTPDA_DEG2RAD Convert degrees to radians

 CALL:        r = ltpda_deg2rad(deg, min, sec)

 INPUTS:      deg - degrees
              min - minutes
              sec - seconds

 OUTPUTS:     r   - radians

 VERSION: $Id: ltpda_deg2rad.m,v 1.5 2007/07/13 12:17:39 ingo Exp $

 HISTORY: 26-01-2007 M Hewitson
              Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function r = ltpda_deg2rad(deg, min, sec)
% LTPDA_DEG2RAD Convert degrees to radians
%
```

```
%%%
```

utp_ltpda_ss_iir2ss

PURPOSE ^

test function for ltpda_ss_miir2ss

SYNOPSIS ^

function result = utp_ltpda_ss_iir2ss

DESCRIPTION ^

```
 test function for ltpda_ss_miir2ss
%%%
```

```
% VERSION: $Id: utp_ltpda_ss_iir2ss.m,v 1.1 2008/02/25 12:43:44 adrien Exp $
%
% HISTORY: 22-01-2008 A Grynagier
% Creation 02-01-2008 A Grynagier
%
% TO DO : more numerical checks with vpa function and jordan block
% decomposition in ltpda_ss_miir2ss
%
%%%
```

```
DMATS = find(subsys,'DMATS');
a = AMAT{1};
b = BMATS{1};
c = CMAT{1};
d = DMATS{1};
A=[0 1; 2 3];
B=[0;1];
C=[2 -1];
D=1;
if (norm(A-[0 1; 2 3]) + norm(B-[0;1]) + norm(C-[2 -1]) + norm(D-1))>1e-14
    result = 0;
end



result = 1;

% display(subsys)
% display(f)

end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY overloads display functionality for miir objects.

SYNOPSIS ^

function txt = display(fs, varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for miir objects.

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

 DESCRIPTION: DISPLAY overloads display functionality for miir objects.

 CALL:        txt  = display(miir)

 INPUT:       miir - IIR filter object

 OUTPUT:      txt  - cell array with strings to display the miir object

 VERSION:     $Id: display.m,v 1.11 2008/02/12 22:46:55 hewitson Exp $

 HISTORY:     11-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a miir object into a string.
- display DISPLAY overloads display functionality for miir objects.
- get GET get miir properties.

This function is called by:

- display DISPLAY overloads display functionality for miir objects.
- set SET set a miir object property.

SOURCE CODE ^

```
function txt = display(fs, varargin)
% DISPLAY overloads display functionality for miir objects.
%
%%%
```

```
  txt{end+1} = sprintf('created: %s', char(f.created));
  txt{end+1} = sprintf('version: %s', f.version);
  txt{end+1} = sprintf('   name: %s', f.name);
  txt{end+1} = ['     fs: ' num2str(f.fs)];
  txt{end+1} = ['  ntaps: ' num2str(f.ntaps)];
  txt{end+1} = ['      a: ' num2str(f.a)];
  txt{end+1} = ['      b: ' num2str(f.b)];
  txt{end+1} = ['   gain: ' num2str(f.gain)];
  txt{end+1} = [' histin: ' num2str(f.histin)];
  txt{end+1} = ['histout: ' num2str(f.histout)];

  txt{end+1} = sprintf('Parameters:');
  txt_plist = display(f.plist);

  for ii = 1:length(txt_plist)
    txt{end+1} = txt_plist{ii};
  end

  txt{end+1} = ' ';

  banner_end(1:length(banner_start)) = '-';
  txt{end+1} = banner_end;

  txt{end+1} = ' ';
  txt{end+1} = ' ';
end

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_dft Linux x86

PURPOSE ^

LTPDA_DFT computes the DFT of a signal at one frequency.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
LTPDA_DFT computes the DFT of a signal at one frequency.

M Hewitson 15-01-08

$Id: ltpda_dft.m,v 1.1 2008/01/15 21:08:47 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
% LTPDA_DFT computes the DFT of a signal at one frequency.
%
% M Hewitson 15-01-08
%
% $Id: ltpda_dft.m,v 1.1 2008/01/15 21:08:47 hewitson Exp $
%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

utp_ltpda_ss_kalman_filter

PURPOSE ^

tests ltpda_simulate_discrete which simulates a discrete state space system

SYNOPSIS ^

function result = utp_ltpda_ss_kalman_filter

DESCRIPTION ^

```
tests ltpda_simulate_discrete which simulates a discrete state space system
%%%
```

```
function result = utp_ltpda_ss_kalman_filter
%tests ltpda_simulate_discrete which simulates a discrete state space system
%%%
```

```
ao_out = ltpda_simulate_discrete(ao_in_list, syst);
%
data_plist = plist('Qs',{0 sigma3^2+mean3^2},'Rs',{sigma2^2+mean2^2
0},'Noise_Names',{'V' 'W'} );

ao_in_list2 = plist('U',ao1);
ao_id = ltpda_ss_kalman_filter(ao_out, ao_in_list2, data_plist, syst);

figure(1)
plot(ao_out)
figure(2)
plot(ao_id)


end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY display a parameter

SYNOPSIS ^

function txt = display(p, varargin)

DESCRIPTION ^

```
 DISPLAY display a parameter

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

 DESCRIPTION: DISPLAY display a parameter

 VERSION: $Id: display.m,v 1.22 2008/02/21 16:48:56 ingo Exp $

 HISTORY: 30-01-2007 M Hewitson
            Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- get GET get parameter properties.

This function is called by:

- set SET set a parameter property.

SOURCE CODE ^

```
function txt = display(p, varargin)
% DISPLAY display a parameter
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: DISPLAY display a parameter
```

```
%
% VERSION: $Id: display.m,v 1.22 2008/02/21 16:48:56 ingo Exp $
%
% HISTORY: 30-01-2007 M Hewitson
%             Creation
%
%%%
```

```
  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];

  %%%%%   Specwin Object   %%%%%
  elseif isa(v, 'specwin');
    txt{end+1} = ['val: ' v.name];
    txt{end+1} = ['     alpha    = ' num2str(v.alpha)];
    txt{end+1} = ['     psll     = ' num2str(v.psll)];
    txt{end+1} = ['     rov      = ' num2str(v.rov)];
    txt{end+1} = ['     nenbw    = ' num2str(v.nenbw)];
    txt{end+1} = ['     w3db     = ' num2str(v.w3db)];
    txt{end+1} = ['     flatness = ' num2str(v.flatness)];

  %%%%%   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 ': ' ltpda_mat2str(v.(field))];
    elseif isobject(v.(field))
      txt{end+1} = ['     ' field ': ' class(v.(field)) '-object'];
    else
      error('### Do not use a [%s] in a cell array', class(v));
    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 '[' ltpda_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
  txt{end+1} = ['val: ' sval];

%%%%%   JAVA   %%%%%
```

```
  elseif isjava(v)
    txt{end+1} = ['val: ' class(v)];

  else
    error('### parameter has unknown type')
  end

  banner_end(1:length(banner)) = '-';
  txt{end+1} = banner_end;

end

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end


```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_disp

PURPOSE ^

LTPDA_DISP display a formatted string to screen.

SYNOPSIS ^

function varargout = ltpda_disp(varargin)

DESCRIPTION ^

```
LTPDA_DISP display a formatted string to screen.

usage:     ltpda_disp(format, args);
       s = ltpda_disp(format, args);

Examples:
     >> ltpda_disp('%s-%d', 'hello', 3);
     >> s = ltpda_disp('%s-%d', 'hello', 3);

M Hewitson 26-02-08

$Id: ltpda_disp.m,v 1.1 2008/02/26 09:30:01 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = ltpda_disp(varargin)

% LTPDA_DISP display a formatted string to screen.
%
% usage:     ltpda_disp(format, args);
%        s = ltpda_disp(format, args);
```

```
%
% Examples:
%      >> ltpda_disp('%s-%d', 'hello', 3);
%      >> s = ltpda_disp('%s-%d', 'hello', 3);
%
% M Hewitson 26-02-08
%
% $Id: ltpda_disp.m,v 1.1 2008/02/26 09:30:01 hewitson Exp $


ALGONAME = mfilename;
VERSION  = '$Id: ltpda_disp.m,v 1.1 2008/02/26 09:30:01 hewitson Exp $';
CATEGORY = 'Internal';

% Check if this is a call for parameters
if nargin == 1
  if  ischar(varargin{1})
    in = char(varargin{1});
    if strcmp(in, 'Params')
      varargout{1} = plist;
      return
    elseif strcmp(in, 'Version')
      varargout{1} = VERSION;
      return
    elseif strcmp(in, 'Category')
      varargout{1} = CATEGORY;
      return
    end
  end
end

% so simple
s = sprintf(varargin{1}, varargin{2:end});
disp(s);

if nargout == 1
  varargout{1} = s;
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

redesign

PURPOSE ^

REDESIGN redesign the input filter to work for the given sample rate.

SYNOPSIS ^

function filt = redesign(filt, fs)

DESCRIPTION ^

```
 REDESIGN redesign the input filter to work for the given sample rate.

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

 DESCRIPTION: REDESIGN redesign the input filter to work for the given sample
              rate.

 CALL:        filt = redesign(filt, fs)

 INPUT:       filt - input (mfir) filter
              fs   - new sample rate

 OUTPUT:      filt - new output filter

 VERSION:     $Id: redesign.m,v 1.4 2008/02/12 22:39:49 hewitson Exp $

 HISTORY:     04-04-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a mfir object into a string.
- get GET get mfir properties.
- mfir MFIR FIR filter object class constructor.
- set SET set a mfir object property.

This function is called by:

SOURCE CODE ^

```
function filt = redesign(filt, fs)
% REDESIGN redesign the input filter to work for the given sample rate.
%
%%%
```

```
    strcmp(type, 'bandpass') ||...
    strcmp(type, 'bandreject')

  disp('   - re-designing standard filter.');
  try
    fpl = pset(fpl, 'fs', fs);
    disp('   - setting new fs.');
  catch
    fpl = append(fpl, 'fs', fs);
    disp('   - appending parameter fs.');
  end
  filtnew = mfir(fpl);
  filtnew = set(filtnew, 'histout', get(filt, 'histout'));
  filt = filtnew;
elseif ~isempty(pzm)

  disp('   - re-designing pzmodel filter.');
  filtnew = mfir(plist([param('pzmodel', pzm) param('fs', fs)]));
  filtnew = set(filtnew, 'histout', get(filt, 'histout'));
  filt = filtnew;

else

  warning('!!! un-recognised input filter type. Can''t redesign.');



end



% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

utp_ltpda_ss_make

PURPOSE ^

tests utp_ltpda_ss_make

SYNOPSIS ^

function result = utp_ltpda_ss_make

DESCRIPTION ^

```
 tests utp_ltpda_ss_make
%%%
```

```
% tests utp_ltpda_ss_make
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_fields2list

PURPOSE ^

LTPDA_FIELDS2LIST splits a string containing fields seperated by ','

SYNOPSIS ^

function fnames = ltpda_fields2list(fields)

DESCRIPTION ^

```
 LTPDA_FIELDS2LIST splits a string containing fields seperated by ','

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

 DESCRIPTION: LTPDA_FIELDS2LIST splits a string containing fields seperated by
              ',' and returns a cell array.

 CALL:        fnames = ltpda_fields2list(fields)

 INPUTS:      string - a string seperated by ','

 OUTPUTS:     fnames - cell array

 EXAMPLE:     >> fields = 'field1, field2, field3, field4';
              >> ltpda_fields2list(fields)
              ans =
                    'field1'    ' field2'    ' field3'    ' field4'

 VERSION: $Id: ltpda_fields2list.m,v 1.5 2007/07/13 12:17:39 ingo Exp $

 HISTORY: 26-01-2007 M Hewitson
              Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function fnames = ltpda_fields2list(fields)
% LTPDA_FIELDS2LIST splits a string containing fields seperated by ','
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: LTPDA_FIELDS2LIST splits a string containing fields seperated by
%              ',' and returns a cell array.
%
% CALL:       fnames = ltpda_fields2list(fields)
%
% INPUTS:     string - a string seperated by ','
%
% OUTPUTS:    fnames - cell array
%
% EXAMPLE:    >> fields = 'field1, field2, field3, field4';
%             >> ltpda_fields2list(fields)
%             ans =
%                   'field1'    ' field2'    ' field3'    ' field4'
%
% VERSION: $Id: ltpda_fields2list.m,v 1.5 2007/07/13 12:17:39 ingo Exp $
%
% HISTORY: 26-01-2007 M Hewitson
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fnames = [];
[f,r] = strtok(fields, ',');
fnames = [fnames  cellstr(f)];
while ~isempty(r)

  [f,r] = strtok(r, ',');
  fnames = [fnames  cellstr(f)];

end


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

redesign

PURPOSE ^

REDESIGN redesign the input filter to work for the given sample rate.

SYNOPSIS ^

function filt = redesign(filt, fs)

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

 VERSION:     $Id: redesign.m,v 1.9 2008/02/12 22:46:55 hewitson Exp $

 HISTORY:     04-04-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a miir object into a string.
- get GET get miir properties.
- miir MIIR IIR filter object class constructor.
- set SET set a miir object property.

This function is called by:

SOURCE CODE ^

```
function filt = redesign(filt, fs)
% REDESIGN redesign the input filter to work for the given sample rate.
%
%%%
```

```
    strcmp(type, 'bandpass') ||...
    strcmp(type, 'bandreject')

  disp('   - re-designing standard filter.');
  try
    fpl = pset(fpl, 'fs', fs);
    disp('   - setting new fs.');
  catch
    fpl = append(fpl, 'fs', fs);
    disp('   - appending parameter fs.');
  end
  filtnew = miir(fpl);
  filtnew = set(filtnew, 'histin', get(filt, 'histin'));
  filtnew = set(filtnew, 'histout', get(filt, 'histout'));
  filt = filtnew;
elseif ~isempty(pzm)

  disp('   - re-designing pzmodel filter.');
  filtnew = miir(plist([param('pzmodel', pzm) param('fs', fs)]));
  filtnew = set(filtnew, 'histin', get(filt, 'histin'));
  filtnew = set(filtnew, 'histout', get(filt, 'histout'));
  filt = filtnew;

else

  warning('!!! un-recognised input filter type. Can''t redesign.');



end



% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

utp_ltpda_ss_modify

PURPOSE ^

tests ltpda_ss_modify

SYNOPSIS ^

function result = utp_ltpda_ss_modify(varargin)

DESCRIPTION ^

```
 tests ltpda_ss_modify
%%%
```

SOURCE CODE ^

```
function result = utp_ltpda_ss_modify(varargin)
% tests ltpda_ss_modify
%%%
```

```
%     ParamSigma = find(syst,'PARAMSIGMAS');
    NbParam = length(ParamNames);
    % input data
%     InputNames = find(syst,'INPUTNAMES');
%     InputSizes = find(syst,'INPUTSIZES');
%     IsUsed = find(syst,'INPUTISUSED');
%     NbInputs = length(InputNames);
    % output data
%     XIsOutput = find(syst,'XISOUTPUT');
%     YIsOutput = find(syst,'YISOUTPUT');

%% modifying system
%     A0 = AMat{1};
%     p=1;
%     for j = 1:NbParam
%         p = p * ParamValue(j);
%         if length(A0)>j
%             A0(j)=A0(j)*p;
%         end
%     end
%     syst = pset(syst, 'AMAT', {A0});



%% setting modification option
    command = floor(2*rand(11,1));
    if NbParam>1&&command(1)
        NOT_USED_PARAM_NAMES = {ParamNames{1}, ParamNames{2}};
        NOT_USED_PARAM_ORDER = [1 0];
    else
        NOT_USED_PARAM_NAMES = {};
        NOT_USED_PARAM_ORDER = [];
    end

    if NbParam>5
        SET_PARAM_NAMES = {ParamNames{3}, ParamNames{4}, ParamNames{5},
ParamNames{6}};
        if command(3)
            SET_PARAM_VALUES_ORDER = [1 1 0 0];
        elseif command(4)
            SET_PARAM_VALUES_ORDER = 'ALL';
        else
            SET_PARAM_VALUES_ORDER = 'NO';
        end
        if command(5)
            SET_PARAM_SIGMAS_ORDER = [1 0 1 0];
        elseif command(6)
            SET_PARAM_SIGMAS_ORDER = 'ALL';
        else
            SET_PARAM_SIGMAS_ORDER = 'NO';
        end
    else
```

```
        SET_PARAM_NAMES = {};
        if command(3)
            SET_PARAM_VALUES_ORDER = [];
        elseif command(4)
            SET_PARAM_VALUES_ORDER = 'ALL';
        else
            SET_PARAM_VALUES_ORDER = 'NO';
        end
        if command(5)
            SET_PARAM_SIGMAS_ORDER = [];
        elseif command(6)
            SET_PARAM_SIGMAS_ORDER = 'ALL';
        else
            SET_PARAM_SIGMAS_ORDER = 'NO';
        end
    end
    SET_PARAM_VALUES = randn(1,length(SET_PARAM_NAMES));
    SET_PARAM_SIGMAS = 0.1*rand(1,length(SET_PARAM_NAMES));

    if NbParam>10
        JAC_PARAM_NAMES = {ParamNames{4}, ParamNames{5}, ParamNames{7},
ParamNames{8}, ParamNames{9}, ParamNames{10}};
        if command(5)
            JAC_PARAM_ORDER = [1 0 1 1 0 0];
        elseif command(6)
            JAC_PARAM_ORDER = 'ALL';
        else
            JAC_PARAM_ORDER = 'NO';
        end
    else
        JAC_PARAM_NAMES = {};
        if command(5)
            JAC_PARAM_ORDER = [];
        elseif command(6)
            JAC_PARAM_ORDER = 'ALL';
        else
            JAC_PARAM_ORDER = 'NO';
        end
    end
    if NbParam>10
        HESS_PARAM_NAMES = {ParamNames{4}, ParamNames{5}, ParamNames{7},
ParamNames{8}, ParamNames{9}, ParamNames{10}};
        if command(7)
            HESS_PARAM_ORDER = [1 0 1 0 1 0];
        elseif command(8)
            HESS_PARAM_ORDER = 'ALL';
        else
            HESS_PARAM_ORDER = 'NO';
        end
    else
        HESS_PARAM_NAMES = {};
        if command(9)
```

```
            HESS_PARAM_ORDER = [];
        elseif command(10)
            HESS_PARAM_ORDER = 'ALL';
        else
            HESS_PARAM_ORDER = 'NO';
        end
    end
    SYMBOLIC_OUTPUT_ALLOWED = command(11);
    options = plist( 'NOT_USED_PARAM_NAMES' , NOT_USED_PARAM_NAMES ,
'NOT_USED_PARAM_ORDER' , NOT_USED_PARAM_ORDER , ...
        'SET_PARAM_NAMES' , SET_PARAM_NAMES , 'SET_PARAM_VALUES' ,
SET_PARAM_VALUES , 'SET_PARAM_SIGMAS' , SET_PARAM_SIGMAS , ...
        'SET_PARAM_VALUES_ORDER', SET_PARAM_VALUES_ORDER ,
'SET_PARAM_SIGMAS_ORDER' , SET_PARAM_SIGMAS_ORDER , ...
        'JAC_PARAM_NAMES' , JAC_PARAM_NAMES ,  'JAC_PARAM_ORDER',
JAC_PARAM_ORDER , ...
        'HESS_PARAM_NAMES' , HESS_PARAM_NAMES ,  'HESS_PARAM_ORDER',
HESS_PARAM_ORDER , ...
        'SYMBOLIC_OUTPUT_ALLOWED', SYMBOLIC_OUTPUT_ALLOWED);

%% launching function
    try
        subsys = ltpda_ss_modify(syst,options);
        display('OK')
    catch
        display('FAIL')
        result = 0;
    end


end

end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY overloads display functionality for pole objects.

SYNOPSIS ^

function txt = display(p, varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for pole objects.

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

 DESCRIPTION: DISPLAY overloads display functionality for pole objects.

 CALL:        txt  = display(pole)

 INPUT:       pole - pole object

 OUTPUT:      txt  - cell array with strings to display the pole object

 VERSION:     $Id: display.m,v 1.5 2008/02/12 21:38:23 hewitson Exp $

 HISTORY:     30-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pole object into a string.
- get GET get pole properties.

This function is called by:

- set SET set a pole property.

SOURCE CODE ^

```
function txt = display(p, varargin)
% DISPLAY overloads display functionality for pole objects.
%
%%%
```

```
0052
0053   % display
0054   if ~isempty(f)
0055     if q > 0.5
0056       txt{end+1} = [ sprintf('%s: %g Hz, Q=%g [', name, f, q) num2str(ri(1))
']'];
0057     else
0058       txt{end+1} = sprintf('%s: %g Hz', name, f);
0059     end
0060   end
0061
0062   banner_end(1:length(banner_start)) = '-';
0063   txt{end+1} = banner_end;
0064
0065 end
0066
0067 if nargout == 0
0068   for ii=1:length(txt)
0069     disp(txt{ii});
0070   end
0071 end
0072
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_filescan

PURPOSE ^

LTPDA_FILESCAN recursively scans the given directory for files that end in 'ext'

SYNOPSIS ^

function files = ltpda_filescan(root_dir, ext)

DESCRIPTION ^

```
 LTPDA_FILESCAN recursively scans the given directory for files that end in 'ext'

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

 DESCRIPTION: LTPDA_FILESCAN recursively scans the given directory for files
              that end in 'ext' and returns a list of the full paths.

 CALL:        files = ltpda_filescan(root_dir, ext)

 INPUTS:      root_dir - directory
              ext      - extension of a file (or cell-array of extensions)

 OUTPUTS:     files    - the found file names

 VERSION: $Id: ltpda_filescan.m,v 1.7 2008/02/26 08:18:28 hewitson Exp $

 HISTORY: 26-01-2007 M Hewitson
              Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

- function ofiles = getfiles(root_dir, iext, ofiles)

SOURCE CODE ^

```
function files = ltpda_filescan(root_dir, ext)
% LTPDA_FILESCAN recursively scans the given directory for files that end in 'ext'
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

utp_ltpda_ss_modify_time_setup

PURPOSE ^

tests ltpda_ss_modify_time_setup with both syntaxic and numerical test.

SYNOPSIS ^

function output = utp_ltpda_ss_modify_time_setup

DESCRIPTION ^

```
 tests ltpda_ss_modify_time_setup with both syntaxic and numerical test.
%%%
```

```
%%%
```

```
        syst4 = ltpda_ss_modify_time_setup(syst3,NewTimeStep);

        NewTimeStep = plist('TIMESTEP',0);
        j = j+1;
        syst5 = ltpda_ss_modify_time_setup(syst4,NewTimeStep);
        display('ok')
    catch
        display(['error at step ' num2str(j)])
        output = 0;
        display('pb')
    end

    %% checking syst2 and sys4 are equal, also for syst0 and syst5
    AMAT2 = find(syst2,'AMAT');
    AMAT4 = find(syst4,'AMAT');
    AMAT0 = find(syst0,'AMAT');
    AMAT5 = find(syst5,'AMAT');
%     BMATS2 = find(syst2,'BMATS');
%     BMATS4 = find(syst4,'BMATS');
%     BMATS0 = find(syst0,'BMATS');
%     BMATS5 = find(syst5,'BMATS');
%     CMAT2 = find(syst2,'CMAT');
%     CMAT4 = find(syst4,'CMAT');
%     CMAT0 = find(syst0,'CMAT');
%     CMAT5 = find(syst5,'CMAT');
    if not( norm(AMAT2{1}-AMAT4{1})<5e-7)
        display('error possible when changing time between discrete and discrete')
        output = 0;
    end
    if not(norm(AMAT0{1}-AMAT5{1})<5e-7)
        display('error possible when changing time between continuous and discrete')
        output = 0;
    end
    if not(norm(AMAT2{1}-AMAT4{1})<1e-13)
        display(['small error when changing time between discrete and discrete :' num2str(norm(AMAT2{1}-AMAT4{1}))])
    end
    if not(norm(AMAT0{1}-AMAT5{1})<1e-13)
        display(['small error when changing time between continuous and discrete : ' num2str(norm(AMAT0{1}-AMAT5{1}))])
    end
end
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY overload terminal display for provenance objects.

SYNOPSIS ^

function txt = display(ps, varargin)

DESCRIPTION ^

```
 DISPLAY overload terminal display for provenance objects.

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

 DESCRIPTION: DISPLAY overload terminal display for provenance objects.

 VERSION: $Id: display.m,v 1.9 2008/02/13 17:11:03 mauro Exp $

 HISTORY: 07-05-2007 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a provenance object into a string.
- get GET get provenance properties.

This function is called by:

- set SET set an provenance property.

SOURCE CODE ^

```
function txt = display(ps, varargin)
% DISPLAY overload terminal display for provenance objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: DISPLAY overload terminal display for provenance objects.
```

```
%
% VERSION: $Id: display.m,v 1.9 2008/02/13 17:11:03 mauro Exp $
%
% HISTORY: 07-05-2007 M Hewitson
%             Creation
%
%%%
```

```
end

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_fq2ri

PURPOSE ^

LTPDA_FQ2RI Convert frequency/Q pole/zero representation into real

SYNOPSIS ^

function [G,ri]= ltpda_fq2ri(f0, Q)

DESCRIPTION ^

```
LTPDA_FQ2RI Convert frequency/Q pole/zero representation into real
and imaginary conjugate pairs.

[G,ri]= ltpda_fq2ri(f0, Q)

M Hewitson 26-01-07

$Id: ltpda_fq2ri.m,v 1.4 2007/01/26 14:03:24 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function [G,ri]= ltpda_fq2ri(f0, Q)

% LTPDA_FQ2RI Convert frequency/Q pole/zero representation into real
% and imaginary conjugate pairs.
%
% [G,ri]= ltpda_fq2ri(f0, Q)
%
% M Hewitson 26-01-07
```

```
%
% $Id: ltpda_fq2ri.m,v 1.4 2007/01/26 14:03:24 hewitson Exp $
%

if(nargin==0)
  disp('usage: [G,ri]= ltpda_fq2ri(f0, Q)')
  return
end

% a = Q - 0.5 ;
% re = 2*pi*f0 / sqrt(1+tan(a)^2);
% im = re * tan(a);

w0 = 2*pi*f0;
re = -w0/(2*Q);
im = w0*sqrt(4*Q*Q-1)/(2*Q);
G = 1/(w0*w0);
ri = complex(re,im);

% END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.11 2008/02/25 16:11:39 hewitson 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 1.11 2008/02/25 16:11:39 hewitson 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');

%-- Retrieve Btn

Bleft   = 10;
Bheight = ALLHEIGHT;
Bwidth  = 100;
Bbottom = Ppos(2) + 10;
pbh = uicontrol(panel,'Style','pushbutton','String','Retrieve',...
  'Position',[Bleft Bbottom Bwidth Bheight ],...
  'Fontsize', fontsize,...
  'Fontweight', 'bold',...
  'Tag', 'retrieveBtn');
set(pbh, 'Callback', {'cb_retrieveBtn', pbh, mainfig});



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',...
  '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,...
  '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,...
  '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 = 'Describe the analysis you did, not the experiment.';
set(sth, 'Tooltipstring', ttstr)

%-----------------------------------------------------------------------
%-- quantity
Tbottom = Tbottom - 20 - Sheight;

sth = uicontrol(panel,'Style','text',...
  'String','Quantity',...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'left',...
  'Position',[Sleft Tbottom Swidth Sheight]);

Theight = ALLHEIGHT;
Tbottom = Tbottom;
sth = uicontrol(panel,'Style','edit',...
  'String','',...
  'Fontsize', fontsize,...
  'BackgroundColor', 'w',...
  'Max', 1,...
  'HorizontalAlignment', 'left',...
  'Position',[Tleft Tbottom Twidth Theight],...
  'Tag', 'quantity');

ttstr = 'The physical quantity that the data represents.';
set(sth, 'Tooltipstring', ttstr)

%-----------------------------------------------------------------------
%-- keywords
Tbottom = Tbottom - 20 - Theight;

sth = uicontrol(panel,'Style','text',...
  'String','Keywords',...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'left',...
  'Position',[Sleft Tbottom Swidth Sheight]);

Extra   = 20;
Theight = ALLHEIGHT + Extra;
Tbottom = Tbottom - Extra;
sth = uicontrol(panel,'Style','edit',...
  'String','',...
  'Fontsize', fontsize,...
  'BackgroundColor', 'w',...
  'Max', 1,...
```

```
  'HorizontalAlignment', 'left',...
  'Position',[Tleft Tbottom Twidth Theight],...
  'Tag', 'keywords');

ttstr = 'A comma-delimeted list of keywords.';
set(sth, 'Tooltipstring', ttstr)

%-----------------------------------------------------------------------
%-- reference ids
Tbottom = Tbottom - Theight;

sth = uicontrol(panel,'Style','text',...
  'String','Reference IDs',...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'left',...
  'Position',[Sleft Tbottom Swidth Sheight]);

Theight = ALLHEIGHT;
Tbottom = Tbottom;
sth = uicontrol(panel,'Style','edit',...
  'String','',...
  'Fontsize', fontsize,...
  'BackgroundColor', 'w',...
  'Max', 1,...
  'HorizontalAlignment', 'left',...
  'Position',[Tleft Tbottom Twidth Theight],...
  'Tag', 'referenceIDs');

ttstr = 'List any object IDs that are relevant to this/these results.';
set(sth, 'Tooltipstring', ttstr)

%-----------------------------------------------------------------------
%-- additional comments
Tbottom = Tbottom - 20 - Theight;

sth = uicontrol(panel,'Style','text',...
  'String','Additional Comments',...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'left',...
  'Position',[Sleft Tbottom Swidth Sheight]);

Extra = 30;
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');

pmh = uicontrol(panel,'Style','popupmenu',...
  'String',{'localhost','130.75.117.67'},...
  '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

utp_ltpda_ss_pz2ss

PURPOSE ^

test conversion from state-space to pole zero model

SYNOPSIS ^

function result = utp_ltpda_ss_pz2ss

DESCRIPTION ^

```
 test conversion from state-space to pole zero model
%%%
```

```
function result = utp_ltpda_ss_pz2ss
% test conversion from state-space to pole zero model
%%%
```

display

PURPOSE ^

DISPLAY overloads display functionality for pzmodel objects.

SYNOPSIS ^

function txt = display(p, varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for pzmodel objects.

%%%
```

```
function txt = display(p, varargin)
% DISPLAY overloads display functionality for pzmodel objects.
%
%%%
```

```
  np    = length(ps);
  nz    = length(zs);

  % display
  txt{end+1} = ['model:    ' name];
  txt{end+1} = ['gain :    ' num2str(g)];
  for j=1:np
    txt{end+1} = [sprintf('pole %03d: ', j) char(ps(j)) ];
  end
  for j=1:nz
    txt{end+1} = [sprintf('zero %03d: ', j) char(zs(j)) ];
  end

  banner_end(1:length(banner)) = '-';
  txt{end+1} = banner_end;

end

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_fq2ri2

PURPOSE ^

LTPDA_FQ2RI2 Convert frequency/Q pole/zero representation into real

SYNOPSIS ^

function ri= ltpda_fq2ri2(f0, Q)

DESCRIPTION ^

```
LTPDA_FQ2RI2 Convert frequency/Q pole/zero representation into real
and imaginary conjugate pairs. Returns [ri conj(ri)]

ri= ltpda_fq2ri2(f0, Q)

M Hewitson 26-01-07


$Id: ltpda_fq2ri2.m,v 1.6 2007/07/11 14:06:39 anneke Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ri= ltpda_fq2ri2(f0, Q)

% LTPDA_FQ2RI2 Convert frequency/Q pole/zero representation into real
% and imaginary conjugate pairs. Returns [ri conj(ri)]
%
% ri= ltpda_fq2ri2(f0, Q)
%
% M Hewitson 26-01-07
```

```
%
%
% $Id: ltpda_fq2ri2.m,v 1.6 2007/07/11 14:06:39 anneke Exp $
%

if(nargin==0)
  disp('usage: ri = fq2ri2(f0, Q)')
  return
elseif(nargin==1)
    ri = (1/(2*pi*f0));
elseif(nargin==2)
    if Q <= 0.5
      disp('working on Q < 0.5') %ri =1/(2*pi*f0);
    else
      w0 = 2*pi*f0;
      G = 1/(w0*w0);
      re = -w0/(2*Q);
      im = w0*sqrt(4*Q*Q-1)/(2*Q);
      tri = complex(re,im);
      ri = G.*[tri conj(tri)]';
    end
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

utp_ltpda_ss_random_generate

PURPOSE ^

utp_ltpda_ss_random_generate subfunction generates a random system

SYNOPSIS ^

function syst = utp_ltpda_ss_random_generate(nSysTot, pSys, nUTot, OutputSizes, InputSizes)

DESCRIPTION ^

```
 utp_ltpda_ss_random_generate subfunction generates a random system
%%%
```

- utp_ltpda_ss_assemble_2 test function for ltpda_ss_assemble - syntaxic
- utp_ltpda_ss_modify tests ltpda_ss_modify
- utp_ltpda_ss_modify_time_setup tests ltpda_ss_modify_time_setup with both syntaxic and numerical test.
- utp_ltpda_ss_ss2iir make is the generic function to decalre or load a subsystem.

SOURCE CODE ^

```
function syst = utp_ltpda_ss_random_generate(nSysTot, pSys, nUTot, OutputSizes, InputSizes)
% utp_ltpda_ss_random_generate subfunction generates a random system
%%%
```

```
end
INPUTISUSED = floor(2*rand(NBINPUTS,1));
%time representation for the matrices, given by the timestep for discrete
%case and by a 0 for a time continuous representation
TIMESTEP = 0;  %to be randomized !!!
%declaration of the symbollic variables TO CHANGE AND LOAD DATA FROM A
%PLIST (WRITE FUNCTION TO *MODIFY* THE PLIST SEPARATELY)
NBPARAMS = floor(10*rand);
PARAMNAMES = cell(NBPARAMS,1);
PARAMVALUES = randn(NBPARAMS,1);
PARAMSIGMAS = rand(NBPARAMS,1);
for i=1:NBPARAMS
    PARAMNAMES{i} = ['Param_Y', num2str(pSys),'_',num2str(i)];
end
%declaring the symbolic variables
for indice=1:length(PARAMNAMES)
    declaration = [  eval(['PARAMNAMES{', num2str(indice),  '}']) , '=sym( ''', eval(['PARAMNAMES{' , num2str(indice), '}']) , ''');'];
    eval(declaration);
end
%Making sure no reserved parameter is used
for i=1:length(PARAMNAMES)
    if ( (isequal(PARAMNAMES{i},'z')==1) || (isequal(PARAMNAMES{i},'t')==1) || (isequal(PARAMNAMES{i},'T')==1) || (isequal(PARAMNAMES{i},'s')==1) )
        msg = ['wrong (reserved for time/frequency) name for variable "', PARAMNAMES{i}, '" in MatModel from file : ', NAME];
        error(msg)
    end
end
%Size of state space
nss = floor(abs(2*randn(1)));
A = randn(nss,nss);
C = rand(OutputSizes(pSys),nss);
AMAT = {A};
CMAT = {C};
BMATS = cell(NBINPUTS,1);
DMATS = cell(NBINPUTS,1);
for i=1:NBINPUTS_exo
    BMATS{i} = rand(nss,InputSizes(InputsexoVals(i)));
    DMATS{i} = rand(OutputSizes(pSys),InputSizes(InputsexoVals(i)));
end
for i=1:NBINPUTS_in
    BMATS{i+NBINPUTS_exo} = rand(nss,OutputSizes(InputsInVals(i)));
    DMATS{i+NBINPUTS_exo} = rand(OutputSizes(pSys),OutputSizes(InputsInVals(i)));
end
INPUTSIZES = zeros(length(BMATS),1);
for i=1:length(BMATS)
    INPUTSIZES(i)= size(BMATS{i},2);
end
XISOUTPUT=floor(2*rand);
YISOUTPUT=floor(2*rand);
XINI = rand(nss,1);
```

```

syst = plist('TYPE', TYPE ,'NAME', NAME ,'TIMESTEP', TIMESTEP , ...
    'XISOUTPUT',XISOUTPUT,'YISOUTPUT',YISOUTPUT,'XINI', XINI , ...
    'PARAMNAMES', PARAMNAMES ,'PARAMVALUES', PARAMVALUES ,'PARAMSIGMAS', PARAMSIGMAS ,...
    'NBINPUTS', NBINPUTS ,'INPUTNAMES', INPUTNAMES ,'INPUTSIZES', INPUTSIZES , 'INPUTISUSED', INPUTISUSED ,...
    'AMAT', AMAT ,'BMATS', BMATS ,'CMAT', CMAT ,'DMATS', DMATS );
%making content tests on the subsystem
ltpda_ss_check(syst);
%erasing useless fields
clear A B C D E F PARAMNAMES PARAMVALUES PARAMSIGMAS
%Saving the workspace in the file named "subsystem_GENERATE_workspaceGenerated".
filename = [mfilename,'_workspaceGenerated'];
save(filename);
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY overloads display functionality for specwin objects.

SYNOPSIS ^

function txt = display(ws, varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for specwin objects.

%%%
```

```
function txt = display(ws, varargin)
% DISPLAY overloads display functionality for specwin objects.
%
%%%
```

```
  txt{end+1} = sprintf('   nenbw: %g', w.nenbw);
  txt{end+1} = sprintf('    w3db: %g', w.w3db);
  txt{end+1} = sprintf('flatness: %g', w.flatness);
  txt{end+1} = sprintf('      ws: %g', w.ws);
  txt{end+1} = sprintf('     ws2: %g', w.ws2);
  txt{end+1} = sprintf('     win: %d', length(w.win));

  txt{end+1} = ' ';

  banner_end(1:length(banner)) = '-';
  txt{end+1} = banner_end;

  txt{end+1} = ' ';
  txt{end+1} = ' ';
end

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_funchash

PURPOSE ^

LTPDA_FUNCHASH compute MD5 hash of an LTPDA function.

SYNOPSIS ^

function h = ltpda_funchash(fcnname)

DESCRIPTION ^

```
 LTPDA_FUNCHASH compute MD5 hash of an LTPDA function.

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

 DESCRIPTION: LTPDA_FUNCHASH compute MD5 hash of an LTPDA function.

 CALL:        h = ltpda_funchash(fcnname)

 INPUTS:      fcnname - An LTPDA function name

 OUTPUTS:     h  - the hash string

 PARAMETERS: None.

 EXAMPLES:

     >> h = ltpda_funchash('ao');

 VERSION:      $Id: ltpda_funchash.m,v 1.1 2007/11/21 13:59:29 hewitson Exp $

 HISTORY: 21-11-07 M Hewitson
            Creation.

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ltpda_hash LTPDA_HASH - Convert an input variable into a message digest using any of

This function is called by:

SOURCE CODE ^

```
function h = ltpda_funchash(fcnname)

% LTPDA_FUNCHASH compute MD5 hash of an LTPDA function.
%
%%%
```

reshapeF

PURPOSE ^

RESHAPEF reshape the frequency vector to match the y vector.

SYNOPSIS ^

function fsd = reshapeF(fsd, varargin)

DESCRIPTION ^

```
 RESHAPEF reshape the frequency vector to match the y vector.

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

 DESCRIPTION: RESHAPEF reshape the frequency vector to match the y vector.

 CALL:        fsd = reshapeF(fsd)

 INPUT:       fsd - frequency-series object

 OUTPUT       fsd - the reshaped frequency-series object

 VERSION:     $Id: reshapeF.m,v 1.5 2008/02/12 22:11:21 hewitson Exp $

 HISTORY:     31-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- fsdata FSDATA frequency-series object class constructor.
- setFreq SETFREQ sets the frequency vector of a fsdata object based on the length of the y vector and the sample rate.

SOURCE CODE ^

```
0001 function fsd = reshapeF(fsd, varargin)
```

```
% RESHAPEF reshape the frequency vector to match the y vector.
%
%%%
```

utp_ltpda_ss_ss2iir

PURPOSE ^

make is the generic function to decalre or load a subsystem.

SYNOPSIS ^

function result = utp_ltpda_ss_ss2iir(varargin)

DESCRIPTION ^

```
 make is the generic function to decalre or load a subsystem.
%%%
```

SOURCE CODE ^

```
function result = utp_ltpda_ss_ss2iir(varargin)
% make is the generic function to decalre or load a subsystem.
%%%
```

```
        display('pb')
    end

end

%% test numerique
A=[0 1; 2 3];
B=[0;1];
C=[2 -1];
D=1;
syst = plist('TYPE', 'SUBSYSTEM' ,'NAME', 'SS1' ,'TIMESTEP', 1 , ...
    'XISOUTPUT',0,'YISOUTPUT',1,'XINI', [0 0]' , ...
    'PARAMNAMES', {} ,'PARAMVALUES', [] ,'PARAMSIGMAS', [] ,...
    'NBINPUTS', 1 ,'INPUTNAMES', {'U'} ,'INPUTSIZES', 1 , 'INPUTISUSED', 1 ,...
    'AMAT', {A} ,'BMATS', {B} ,'CMAT', {C} ,'DMATS', {D} );
ltpda_ss_check(syst)
f = ltpda_ss_ss2iir(syst);
fa = [1 -4 0];
fb = [1 -3 -2];
if (norm(fa-f{1}.a) + norm(fb-f{1}.b))>1e-14
    result = 0;
end
display(result)

end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY overloads display functionality for time objects.

SYNOPSIS ^

function txt = display(tt, varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for time objects.

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

 DESCRIPTION: DISPLAY overloads display functionality for time objects.

 CALL:        txt  = display(time)

 INPUT:       time - time object

 OUTPUT:      txt  - cell array with strings to display the time object

 VERSION:     $Id: display.m,v 1.9 2008/02/13 16:31:26 mauro Exp $

 HISTORY:     23-07-2007 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a time object into a string.

This function is called by:

- set SET set time properties.

SOURCE CODE ^

```
0001 function txt = display(tt, varargin)
```

```
% DISPLAY overloads display functionality for time objects.
%
%%%
```

```
      max_length = length(fields{jj});
    end
  end

  for jj = 1:length(fields)
    field = fields{jj};

    str_field = [];
    str_field(1:max_length-length(field)) = ' ';
    str_field = [field str_field];

    % Add before the fields 'created' and 'version' a blank line
    if strcmp(field, 'created')
      txt{end+1} = ' ';
    end

    % Display: Number
    if isnumeric(t.(field))
      txt{end+1} = sprintf ('%s: %s',str_field, num2str(t.(field)));

    % Display: Strings
    elseif ischar(t.(field))
      txt{end+1} = sprintf ('%s: %s',str_field, t.(field));

    % Display: Logicals
    elseif islogical(t.(field))
      if t.(field) == true
        txt{end+1} = sprintf ('%s: true',str_field);
      else
        txt{end+1} = sprintf ('%s: false',str_field);
      end

    % Display: Objects
    elseif isobject(t.(field))
      if isa(t.(field), 'timeformat')
        txt{end+1} = sprintf ('%s: %s',str_field, char(t.(field)));
      else
        txt{end+1} = sprintf ('%s: %s class',str_field, class(t.(field)));
      end

    % Display: Java Objects
    elseif isjava(t.(field))
      if strcmp(class(t.(field)), 'sun.util.calendar.ZoneInfo')
        EE = java.text.SimpleDateFormat('Z');
        EE.setTimeZone(t.(field))
        zone_id = t.(field).getID;
        if strcmp(zone_id, 'UTC')
          txt{end+1} = sprintf ('%s: %s',str_field, char(t.(field).getID));
        else
          txt{end+1} = sprintf ('%s: %s (UTC%s)',str_field, char(t.(field).
getID), char(EE.format(t.utc_epoch_milli)));
        end
```

```
      else
        error ('### Unknown java class %s', class(t.(field)))
      end

    else
      error ('### Please define the output for this property %s', field)
    end

  end


  e_banner(1:length(b_banner)) = '-';
  txt{end+1} = e_banner;

  txt{end+1} = ' ';
  txt{end+1} = ' ';

end

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end


```

Generated on Tue 26-Feb-2008 10:52:52 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.

%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()
- function plf = getlpFilter(x)
- function phf = gethpFilter(x)

SOURCE CODE ^

```
function varargout=ltpda_gapfilling(varargin)
% LTPDA_GAPFILLING fills possible gaps in data.
%
%%%
```

```
ALGONAME = mfilename;
VERSION  = '$Id: ltpda_gapfilling.m,v 1.2 2008/01/29 10:55:27 josep Exp $';
CATEGORY = 'SIGPROC';

% check if this is a call for parameters
if nargin == 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

% Capture input variables names
invars = {};
as     = [];
ps     = [];
for j=1:nargin
   invars = [invars cellstr(inputname(j))];
   if isa(varargin{j}, 'ao')
     as = [as varargin{j}];
   end
   if isa(varargin{j}, 'plist')
     ps = [ps varargin{j}];
   end
end

if length(as)~=2
   error('only two analysis objects are needed!')
end

% check plist
if isempty(ps)
  pl = getDefaultPL();
else
  pl = combine(ps, getDefaultPL);
end

% check input parameters
method = find(pl, 'method'); % method definition: linear or 'spline'
addnoise = find(pl, 'addnoise'); % decide whether add noise or not in
                                 % the filled 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 ~strcmp(a1.xunits,a2.xunits)
          error('!!! Data has different X units !!!');
end

if ~strcmp(a1.yunits,a2.yunits)
          error('!!! Data has different Y units !!!');
end

a1_length = length(a1.x);
a2_length = length(a2.x);
gaptime   = a2.x(1) - a1.x(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.y)>10 && length(a2.y)>10
    dy = mean(a2.y(1:10))-mean(a1.y(a1_length-10:a1_length));

    filling_data = (dy/gaptime)*t + mean(a1.y(a1_length-10:a1_length));
    filling_time = (1:1:gapn)/a1.fs + a1.x(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.y)>1000 && length(a2.y)>1000

    % derivatives of the input data are calculated
    da1 = diff(a1.y(1:100:a1_length))*(a1.fs/100);
    da1 = tsdata(da1, a1.fs/100);
    da1 = ao(da1);

    da2 = diff(a2.y(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.y(length(da1filtered.data.x)...
        -10:length(da1filtered.data.x)));
    d = mean(a1.y(length(a1.y)-10:length(a1.y)));

    a=(2*d+(c+mean(da2filtered.data.y(1:10)))...
        *gaptime-2*mean(a2.y(1:10)))/(gaptime.^3);
    b=-(3*d+2*c*gaptime+mean(da2filtered.data.y(1:10))...
        *gaptime-3*mean(a2.y(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.x(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.y, 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.y;

```

```
end

% join data
filling_data = [a1.y; filling_data'; a2.y];
filling_time = [a1.x; filling_time'; a2.x];

% create new output tsdata
ts = tsdata(filling_time, filling_data)
ts = set(ts, 'name', sprintf('filled data between %s and %s', as(1).name, as(2).name));
ts = set(ts, 'yunits', a1.yunits);
ts = set(ts, 'xunits', a1.xunits);

% create new output history
histin = [as(1).hist as(2).hist];
h = history(ALGONAME, VERSION, pl, histin);
h = set(h, 'invars', invars);

% make output analysis object
b = ao(ts, h);

b = setnh(b, 'name', sprintf('gapfilling(%s, %s)', as(1).name, as(2).name));

varargout{1} = b;

% -------------------------------------
% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('method', 'linear'));
plo = append(plo, param('addnoise', 'no'));

% Get low-pass filter
function plf = getlpFilter(x)

plf = plist();
plf = append(plf, param('gain', 1));
plf = append(plf, param('ripple', 0.5));
plf = append(plf, param('type', 'lowpass'));
plf = append(plf, param('order', 2));
plf = append(plf, param('fs', x));
plf = append(plf, param('fc', 0.1/100));

% Get high-pass filter
function phf = gethpFilter(x)

phf = plist();
phf = append(phf, param('gain', 1));
phf = append(phf, param('ripple', 0.5));
phf = append(phf, param('type', 'highpass'));
phf = append(phf, param('order', 2));
```

```
phf = append(phf, param('fs', x));
phf = append(phf, param('fc', 0.1/100));
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

reshapeT

PURPOSE ^

RESHAPET reshape the time-vector to match the x vector.

SYNOPSIS ^

function ts = reshapeT(ts, varargin)

DESCRIPTION ^

```
 RESHAPET reshape the time-vector to match the x vector.

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

 DESCRIPTION: RESHAPET reshape the time-vector to match the x vector.

 CALL:        tsdata = reshape(tsdata)

 VERSION:     $Id: reshapeT.m,v 1.5 2008/02/13 11:39:35 mauro Exp $

 HISTORY:     31-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- setTime SETTIME sets the time-vector of a tsdata object based on the length of the x vector and the sample rate.
- tsdata TSDATA time-series object class constructor.

SOURCE CODE ^

```
0001 function ts = reshapeT(ts, varargin)
0002 % RESHAPET reshape the time-vector to match the x vector.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
```

```
% DESCRIPTION: RESHAPET reshape the time-vector to match the x vector.
%
% CALL:        tsdata = reshape(tsdata)
%
% VERSION:     $Id: reshapeT.m,v 1.5 2008/02/13 11:39:35 mauro Exp $
%
% HISTORY:     31-01-2007 M Hewitson
%                 Creation
%
%%%
```

display

PURPOSE ^

DISPLAY overloads display functionality for timeformat objects.

SYNOPSIS ^

function txt = display(tt, varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for timeformat objects.

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

 DESCRIPTION: DISPLAY overloads display functionality for timeformat objects.

 CALL:        txt = display(tf)

 INPUT:       tf  - timeformat object

 OUTPUT:      txt - cell array with strings to display the timeformat object

 VERSION:     $Id: display.m,v 1.5 2008/02/13 16:54:14 mauro Exp $

 HISTORY:     23-07-2007 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a timeformat object into a string.

This function is called by:

- set SET set timeformat properties.

SOURCE CODE ^

```
0001 function txt = display(tt, varargin)
```

```
% DISPLAY overloads display functionality for timeformat objects.
%
%%%
```

```
      max_length = length(fields{jj});
    end
  end

  for jj = 1:length(fields)
    field = fields{jj};

    str_field = [];
    str_field(1:max_length-length(field)) = ' ';
    str_field = [field str_field];

    % Display: Number
    if isnumeric(t.(field))
      txt{end+1} = sprintf ('%s: %s',str_field, num2str(t.(field)));

    % Display: Strings
    elseif ischar(t.(field))
      txt{end+1} = sprintf ('%s: %s',str_field, t.(field));

    % Display: Logicals
    elseif islogical(t.(field))
      if t.(field) == true
        txt{end+1} = sprintf ('%s: true',str_field);
      else
        txt{end+1} = sprintf ('%s: false',str_field);
      end

    % Display: Objects
    elseif isobject(t.(field))
      txt{end+1} = sprintf ('%s: %s class',str_field, class(t.(field)));

    % Display: Java Objects
    elseif isjava(t.(field))
        error ('### Unknown java class %s', class(t.(field)))

    else
      error ('### Please define the output for this property %s', field)
    end

  end


  e_banner(1:length(b_banner)) = '-';
  txt{end+1} = e_banner;

  txt{end+1} = ' ';
  txt{end+1} = ' ';

end

if nargout == 0
  for ii=1:length(txt)
```

```
    disp(txt{ii});
  end
end


```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_getAOsInTimeSpan

PURPOSE ^

LTPDA_GETAOSINTIMESPAN performs high-level queries to retrieve AOs from an LTPDA repository.

SYNOPSIS ^

function varargout = ltpda_getAOsInTimeSpan(varargin)

DESCRIPTION ^

```
 LTPDA_GETAOSINTIMESPAN performs high-level queries to retrieve AOs from an LTPDA
repository.

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

 DESCRIPTION: LTPDA_GETAOSINTIMESPAN performs high-level queries to retrieve AOs
              from an LTPDA repository. The AOs for that channel in that time-span
              are joined together and then split on the requested
              interval. The resulting AO covers the requested interval,
              assuming that the repositoy contains a contiguos set of AOs
              spanning the requested time.

 CALL:        b = ltpda_getAOsInTimeSpan(pl)

 INPUTS:      pl - a plist containing parameters as described below

 OUTPUTS:     b  - a vector of AOs resulting from the query

 PARAMETERS:

     choose from
              'hostname'       - hostname/IP address of LTPDA Repository
              'database'       - the name of the database to query
          or
              'conn'           - a database connection object

    in addition, we need
              'channels'       - a cell array of AO names
              'timespan'       - a timespan object to specify the
                                 time-interval of interest

 The timespan is applied start <= data < end.

 OPTIONAL PARAMETERS:
```

```
   further restrict the search with the following conditions

              'username'   - specify the username of the person who
                             submitted the AO
              'submitted'  - specify the date the AO was submitted

EXAMPLES:

1) Find all AOs which are called 'ChannelX' submitted after 1st May 2007

   pl = plist('hostname', 'localhost', 'database', 'ltpda_test', ...
              'channels', {'channelX'}, ...
              'timespan', timespan('2007-05-01 00:00:00', '3000-01-01 12:34:50', ...
              );

   a = ltpda_getAOsInTimeSpan(pl);

2) Find all AOs for ChannelX and ChannelY on 1st May 2007 submitted by
   hewitson

   pl = plist('hostname', 'localhost', 'database', 'ltpda_test', ...
              'channels', {'channelX'}, ...
              'timespan', timespan('2007-05-01 00:00:00', '2007-05-02 00:00:00', ...
              'username', 'hewitson');

   a = ltpda_getAOsInTimeSpan(pl);

VERSION:     $Id: ltpda_getAOsInTimeSpan.m,v 1.1 2008/02/25 19:47:22 hewitson Exp $

The following call returns a parameter list object that contains the
default parameter values:

>> pl = ltpda_getAOsInTimeSpan('Params')

The following call returns a string that contains the routine CVS version:

>> version = getAOsInTimeSpan('Version')

The following call returns a string that contains the routine category:

>> category = ltpda_getAOsInTimeSpan('Category')

HISTORY: 25-02-08 M Hewitson
            Creation

%%%
```

This function calls:

- ltpda_dbquery LTPDA_DBQUERY query an AO repository database.
- ltpda_obj_retrieve LTPDA_OBJ_RETRIEVE retrieves a collection of objects from an LTPDA repository.
- mysql_connect MYSQL_CONNECT connects to an AO repository and returns the connection object.

This function is called by:

SUBFUNCTIONS ^

- function objsOut = hlq(conn, channels, ts)
- function pl = getDefaultPlist(varargin)

SOURCE CODE ^

```
function varargout = ltpda_getAOsInTimeSpan(varargin)

% LTPDA_GETAOSINTIMESPAN performs high-level queries to retrieve AOs from an LTPDA repository.
%
%%%
```

```
% The timespan is applied start <= data < end.
%
% OPTIONAL PARAMETERS:
%
%    further restrict the search with the following conditions
%
%               'username'   - specify the username of the person who
%                              submitted the AO
%               'submitted'  - specify the date the AO was submitted
%
% EXAMPLES:
%
% 1) Find all AOs which are called 'ChannelX' submitted after 1st May 2007
%
%    pl = plist('hostname', 'localhost', 'database', 'ltpda_test', ...
%               'channels', {'channelX'}, ...
%               'timespan', timespan('2007-05-01 00:00:00', '3000-01-01 12:34:50', ...
%               );
%
%    a = ltpda_getAOsInTimeSpan(pl);
%
% 2) Find all AOs for ChannelX and ChannelY on 1st May 2007 submitted by
%    hewitson
%
%    pl = plist('hostname', 'localhost', 'database', 'ltpda_test', ...
%               'channels', {'channelX'}, ...
%               'timespan', timespan('2007-05-01 00:00:00', '2007-05-02 00:00:00', ...
%               'username', 'hewitson');
%
%    a = ltpda_getAOsInTimeSpan(pl);
%
% VERSION:     $Id: ltpda_getAOsInTimeSpan.m,v 1.1 2008/02/25 19:47:22 hewitson Exp $
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = ltpda_getAOsInTimeSpan('Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = getAOsInTimeSpan('Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = ltpda_getAOsInTimeSpan('Category')
%
% HISTORY: 25-02-08 M Hewitson
%             Creation
%
%%%
```

```
ALGONAME = mfilename;
VERSION  = '$Id: ltpda_getAOsInTimeSpan.m,v 1.1 2008/02/25 19:47:22 hewitson Exp $';
CATEGORY = 'Constructor';

% Check if this is a call for parameters
if nargin == 1
  if ischar(varargin{1})
    in = char(varargin{1});
    if strcmp(in, 'Params')
      varargout{1} = getDefaultPlist();
      return
    elseif strcmp(in, 'Version')
      varargout{1} = VERSION;
      return
    elseif strcmp(in, 'Category')
      varargout{1} = CATEGORY;
      return
    end
  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);

% Process the input plist
conn     = find(pl, 'conn');
hostname = find(pl, 'hostname');
database = find(pl, 'database');
channels = find(pl, 'channels');
tspan    = find(pl, 'timespan');

% Check inputs
if isempty(hostname) || ~ischar(hostname)
  if isempty(conn) || ~isa(conn, 'database')
    error(sprintf('### Plist should contain ''hostname'' and ''database'' parameters,\n or a ''conn'' parameter.'));
  end
end
if isempty(database) || ~ischar(database)
  if isempty(conn) || ~isa(conn, 'database')
    error(sprintf('### Plist should contain ''hostname'' and ''database'' parameters,\n or a ''conn'' parameter.'));
  end
end
if isempty(channels)
  error('### Plist should contain a ''channels'' parameter.');
end
```

```
if isempty(tspan) || ~isa(tspan, 'timespan')
  error('### Plist should contain a ''timespan'' parameter.');
end

% Check the channels value is a cell
if ~iscell(channels)
  channels = {channels};
end

% Get a connection to the database
iConnected = 0;
if ~isa(conn, 'database')
  % then we connect
  try
    conn = mysql_connect(hostname, database);
    iConnected = 1;
  catch
    error('### Failed to connect to server: %s/%s', hostname, database);
  end
end

% Collect the data
try
  aout =  hlq(conn, channels, tspan);
catch
  % Do we need to close database connection?
  if iConnected
    close(conn);
  end
  error('### Error retrieving objects.');
end

% Do we need to close database connection?
if iConnected
  close(conn);
end

% set outputs
varargout{1} = aout;

%-----------------------------------------------------------------------------
% A prototype for the kind of high-level query functions we might need
function objsOut = hlq(conn, channels, ts)

% prepare output
objsOut = [];

% Loop over the channels
for j=1:length(channels)

  % this channel
  channel = channels{j};
```

```

  % the query
  q =      ['select objmeta.obj_id from objmeta,ao,tsdata ' ...
            'where objmeta.obj_id=ao.obj_id ' ...
            'and ao.data_id=tsdata.id ' ...
    sprintf('and objmeta.name=''%s''', channel) ...
    sprintf('and tsdata.t0+INTERVAL tsdata.nsecs SECOND  >= ''%s''', ts.start.time_str) ...
    sprintf('and tsdata.t0 <= ''%s''', ts.end.time_str)];

  % execute query
  info = ltpda_dbquery(conn, q);

  % collect objects
  objs = ltpda_obj_retrieve(conn, [info{:}]);

  % join these up
  ojn = join([objs{:}]);

  % split out the bit we want
  os = split(ojn, plist('split_type', 'interval', 'timespan', ts));

  % add to outputs
  objsOut = [objsOut os];
end % end loop over channels

%-----------------------------------------------------------------------
% Returns the default plist
function pl = getDefaultPlist(varargin)

pl = plist('hostname', 'localhost', ...
           'database', 'ltpda_test',  ...
           'channels', {'chan1', 'chan2'}, ...
           'timespan', timespan('1970-01-01 00:00:00', '2010-10-12 12:45:32') ...
           );


% END

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

reshapeX

PURPOSE ^

RESHAPEX reshape the X to match the Y vector.

SYNOPSIS ^

function xy = reshapeX(xy, varargin)

DESCRIPTION ^

```
 RESHAPEX reshape the X to match the Y vector.

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

 DESCRIPTION: RESHAPEX reshape the X to match the Y vector.

 CALL:        xydata = reshape(xydata)

 VERSION:     $Id: reshapeX.m,v 1.5 2008/02/13 12:57:47 mauro Exp $

 HISTORY:     31-01-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function xy = reshapeX(xy, varargin)
% RESHAPEX reshape the X to match the Y vector.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: RESHAPEX reshape the X to match the Y vector.
%
% CALL:        xydata = reshape(xydata)
%
```

```
% VERSION:     $Id: reshapeX.m,v 1.5 2008/02/13 12:57:47 mauro Exp $
%
% HISTORY:     31-01-07 M Hewitson
%                 Creation
%
%%%
```

display

PURPOSE ^

DISPLAY overloads display functionality for timespan objects.

SYNOPSIS ^

function txt = display(t_span, varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for timespan objects.

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

 DESCRIPTION: DISPLAY overloads display functionality for timespan objects.

 CALL:        txt = display(ts)

 INPUT:       ts  - timespan object

 OUTPUT:      txt - cell array with strings to display the timespan object

 VERSION:     $Id: display.m,v 1.6 2008/02/13 16:43:13 mauro Exp $

 HISTORY:     23-07-2007 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a timespan object into a string.

This function is called by:

- set SET set timespan properties.

SOURCE CODE ^

```
0001 function txt = display(t_span, varargin)
```

```
% DISPLAY overloads display functionality for timespan objects.
%
%%%
```

```
      max_length = length(fields{jj});
    end
  end

  for jj = 1:length(fields)
    field = fields{jj};

    str_field = [];
    str_field(1:max_length-length(field)) = ' ';
    str_field = [field str_field];

    % Add before the fields 'created' and 'version' a blank line
    if strcmp(field, 'created')
      txt{end+1} = ' ';
    end

    % Display: Number
    if isnumeric(t.(field))
      txt{end+1} = sprintf ('%s: %s',str_field, num2str(t.(field)));

    % Display: Strings
    elseif ischar(t.(field))
      txt{end+1} = sprintf ('%s: %s',str_field, t.(field));

    % Display: Logicals
    elseif islogical(t.(field))
      if t.(field) == true
        txt{end+1} = sprintf ('%s: true',str_field);
      else
        txt{end+1} = sprintf ('%s: false',str_field);
      end

    % Display: Objects
    elseif isobject(t.(field))
      if isa(t.(field), 'time') || isa(t.(field), 'timeformat')
        txt{end+1} = sprintf ('%s: %s',str_field, char(t.(field)));
      else
        txt{end+1} = sprintf ('%s: %s class',str_field, class(t.(field)));
      end

    % Display: Java Objects
    elseif isjava(t.(field))
      if strcmp(class(t.(field)), 'sun.util.calendar.ZoneInfo')
        EE = java.text.SimpleDateFormat('Z');
        EE.setTimeZone(t.(field))
        zone_id = t.(field).getID;
        if strcmp(zone_id, 'UTC')
          txt{end+1} = sprintf ('%s: %s',str_field, char(t.(field).getID));
        else
          txt{end+1} = sprintf ('%s: %s (UTC%s)',str_field, char(t.(field).
getID), char(EE.format(t_span.start.utc_epoch_milli)));
        end
```

```
      else
        error ('### Unknown java class %s', class(t.(field)))
      end

    else
      error ('### Please define the output for this property %s', field)
    end

  end


  e_banner(1:length(b_banner)) = '-';
  txt{end+1} = e_banner;

  txt{end+1} = ' ';
  txt{end+1} = ' ';

end

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_getmax

PURPOSE ^

LTPDA_GETMAX get the index and value of the maximum value in an array.

SYNOPSIS ^

function [index, max] = ltpda_getmax(a)

DESCRIPTION ^

```
 LTPDA_GETMAX get the index and value of the maximum value in an array.

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

 DESCRIPTION: LTPDA_GETMAX get the index and value of the maximum value in an array

 CALL:        [index, max] = ltpda_getmax(a)

 INPUTS:      a     - vector od double

 OUTPUTS:     index - index of the maximum
              max   - value of the maximum

 VERSION: $Id: ltpda_getmax.m,v 1.5 2007/07/13 12:17:39 ingo Exp $

 HISTORY: 26-01-2007 M Hewitson
              Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function [index, max] = ltpda_getmax(a)
% LTPDA_GETMAX get the index and value of the maximum value in an array.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%
% DESCRIPTION: LTPDA_GETMAX get the index and value of the maximum value in an array
%
% CALL:       [index, max] = ltpda_getmax(a)
%
% INPUTS:     a     - vector od double
%
% OUTPUTS:    index - index of the maximum
%             max   - value of the maximum
%
% VERSION: $Id: ltpda_getmax.m,v 1.5 2007/07/13 12:17:39 ingo Exp $
%
% HISTORY: 26-01-2007 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.

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

 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, pl);
                  resp(filt, pl);

 PARAMETERS: 'f1'      -  the start frequency
             'f2'      -  the stop frequency
             'nf'      -  number of evaluation points
             'scale'   - spacing of frequencies: 'lin' or 'log'
        or
             'f'       -  a vector of frequency values

 OUTPUTS:    a       -  an analysis object

 The response is calculated as:

          H(s) = gain * (a0 + a1*exp(s/fs) + ... + an*exp((n-1)s/fs))

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

 The following call returns a cell array that contains the default parameter
 sets names:
```

```
>> sets = resp(mfir, 'Params')

The following call returns a parameter list object that contains the
default parameters for the jj-th set:

>> pl = resp(mfir, 'Params', sets{jj})

The following call returns a string that contains the routine CVS version:

>> version = resp(mfir, 'Version')

The following call returns a string that contains the routine category:

>> category = resp(mfir, 'Category')


VERSION: $Id: resp.m,v 1.12 2008/02/24 21:43:59 mauro Exp $

HISTORY: 27-08-02 M Hewitson
            Creation

%%%
```

```
%              If no outputs are specified, the frequency-series is plotted as
%              mag/deg.
%
% CALL:        a = resp(filt);
%              a = resp(filt, pl);
%                  resp(filt, pl);
%
% PARAMETERS: 'f1'      -  the start frequency
%             'f2'      -  the stop frequency
%             'nf'      -  number of evaluation points
%             'scale'   - spacing of frequencies: 'lin' or 'log'
%        or
%             'f'       -  a vector of frequency values
%
% OUTPUTS:    a       -  an analysis object
%
% The response is calculated as:
%
%          H(s) = gain * (a0 + a1*exp(s/fs) + ... + an*exp((n-1)s/fs))
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% The following call returns a cell array that contains the default parameter
% sets names:
%
% >> sets = resp(mfir, 'Params')
%
% The following call returns a parameter list object that contains the
% default parameters for the jj-th set:
%
% >> pl = resp(mfir, 'Params', sets{jj})
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = resp(mfir, 'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = resp(mfir, 'Category')
%
%
% VERSION: $Id: resp.m,v 1.12 2008/02/24 21:43:59 mauro Exp $
%
% HISTORY: 27-08-02 M Hewitson
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ALGONAME = mfilename;
VERSION  = '$Id: resp.m,v 1.12 2008/02/24 21:43:59 mauro Exp $';
CATEGORY = 'Signal Processing';

```

```
%% 'Params', 'Version' and 'Category' Call
if (nargin == 2 || nargin == 3) && ...
    isa(varargin{1}, 'mfir') && ...
    ischar(varargin{2})
  in = char(varargin{2});
  if strcmp(in, 'Params')
    if nargin == 2
      varargout{1} = getDefaultPL();
    else
      varargout{1} = getDefaultPL(varargin{3});
    end
    return
  elseif strcmp(in, 'Version')
    varargout{1} = VERSION;
    return
  elseif strcmp(in, 'Category')
    varargout{1} = CATEGORY;
    return
  end
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, getDefaultPL('List'));
else
  pl = combine(pl, getDefaultPL('Range'));
end


% fill parameter list
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');
```

```
  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
  if size(f,1) > 1
    f = f.';
  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
dresp = num .* filt.gain;  % apply the gain

% 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;
% dresp = dresp .* filt.g;  % apply the gain

% mag   = 20*log10(abs(dresp));
% phase = angle(dresp)*180/pi;


%% Create an analysis object

% create new output fsdata
```

```
fs = fsdata(f, dresp, filt.fs);
fs = set(fs, 'name', sprintf('resp(%s)', filt.name));

% create new output history
h = history(ALGONAME, VERSION, plo);

% make output analysis object
b = ao(fs, h);

% set name
b = setnh(b, 'name', sprintf('resp(%s)', filt.name));

% Outputs
if nargout == 0
  iplot(b)
end

if nargout == 1
  varargout{1} = b;
end
if nargout > 1
  error('incorrect output arguments');
end


%%%
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY overloads display functionality for tsdata objects.

SYNOPSIS ^

function txt = display(tsdatas, varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for tsdata objects.

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

 DESCRIPTION: DISPLAY overloads display functionality for tsdata objects.

 CALL:        txt    = display(tsdata)

 INPUT:       tsdata - tsdta object

 OUTPUT:      txt    - cell array with strings to display the tsdata object

 VERSION:     $Id: display.m,v 1.10 2008/02/13 11:39:35 mauro Exp $

 HISTORY:     31-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- get GET get tsdata properties.

This function is called by:

- set SET sets a tsdata property.

SOURCE CODE ^

```
function txt = display(tsdatas, varargin)
% DISPLAY overloads display functionality for tsdata objects.
%
%%%
```

```
  yinfo = whos('ydata');

  txt{end+1} = sprintf('  name:  %s', get(ts, 'name'));
  txt{end+1} = sprintf('    fs:  %g', get(ts, 'fs'));
  txt{end+1} = sprintf('     x:  [%d %d], %s', xinfo.size(1), xinfo.size(2), xinfo.class);
  txt{end+1} = sprintf('     y:  [%d %d], %s', yinfo.size(1), yinfo.size(2), yinfo.class);
  txt{end+1} = sprintf('xunits:  %s', get(ts, 'xunits'));
  txt{end+1} = sprintf('yunits:  %s', get(ts, 'yunits'));
  txt{end+1} = sprintf(' nsecs:  %d', get(ts, 'nsecs'));
  txt{end+1} = sprintf('    t0:  %s', char(get(ts, 't0')));

  banner_end(1:length(banner)) = '-';
  txt{end+1} = banner_end;

  txt{end+1} = ' ';
  txt{end+1} = ' ';
end

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_gps2utc

PURPOSE ^

LTPDA_GPS2UTC converts GPS seconds to UTC time.

SYNOPSIS ^

function UTC_time=ltpda_gps2utc(GPS_time)

DESCRIPTION ^

```
 LTPDA_GPS2UTC converts GPS seconds to UTC time.

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

 DESCRIPTION: LTPDA_GPS2UTC converts GPS seconds to UTC time.

 CALL:        UTC_time=ltpda_GPS2UTC(GPS_time)

 FORMAT:      UTC time format: 'yyy-mm-dd- HH:MM:SS'
              GPS time format: Seconds since 6. January 1980

    EXAMPLES:    GPS_time=ltpda_GPS2UTC(711129613)
                 GPS_time='2002-07-19 16:00:00'

 VERSION:     $Id: ltpda_gps2utc.m,v 1.4 2007/07/27 09:31:22 ingo Exp $

 HISTORY:     xx-xx-xxxx Karsten Koetter
                 Creation
              24-01-2007 M Hewitson.
                 Maintained

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function UTC_time=ltpda_gps2utc(GPS_time)
```

```
% LTPDA_GPS2UTC converts GPS seconds to UTC time.
%
%%%
```

resp

PURPOSE ^

RESP Make a frequency response of the filter.

SYNOPSIS ^

function [varargout] = resp(varargin)

DESCRIPTION ^

```
 RESP Make a frequency response of the filter.

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

 DESCRIPTION: RESP Make a frequency response of the filter.
              The input filter should be an miir object.

              The response is returned as a frequency-series in an
              analysis object.

              If no outputs are specified, the frequency-series is plotted as
              mag/deg.

 CALL:        a = resp(filt, pl)
                  resp(filt, pl);

 PARAMETERS:  'f1'      -  the start frequency
              'f2'      -  the stop frequency
              'nf'      -  number of evaluation points
              'scale'   - spacing of frequencies: 'lin' or 'log'
         or
              'f'       -  a vector of frequency values

 OUTPUTS:
              a      -  an analysis object

 The response is calculated as:

                          a0 + a1*exp(s/fs) + ... + an*exp((n-1)s/fs)
          H(s) = gain * ---------------------------------------------
                          b0 + b1*exp(s/fs) + ... + bm*exp((m-1)s/fs)

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

```
The following call returns a cell array that contains the default parameter
sets names:

>> sets = resp(miir, 'Params')

The following call returns a parameter list object that contains the
default parameters for the jj-th set:

>> pl = resp(miir, 'Params', sets{jj})

The following call returns a string that contains the routine CVS version:

>> version = resp(miir, 'Version')

The following call returns a string that contains the routine category:

>> category = resp(miir, 'Category')


VERSION: $Id: resp.m,v 1.10 2008/02/24 21:43:59 mauro Exp $

NOTE:    Some of the code below is taken from Mathworks's treeplot.m

HISTORY: 27-08-02 M Hewitson
            Creation

%%%
```

```
%               The input filter should be an miir object.
%
%               The response is returned as a frequency-series in an
%               analysis object.
%
%               If no outputs are specified, the frequency-series is plotted as
%               mag/deg.
%
% CALL:         a = resp(filt, pl)
%                   resp(filt, pl);
%
% PARAMETERS:   'f1'      -  the start frequency
%               'f2'      -  the stop frequency
%               'nf'      -  number of evaluation points
%               'scale'   - spacing of frequencies: 'lin' or 'log'
%          or
%               'f'       -  a vector of frequency values
%
% OUTPUTS:
%               a      -  an analysis object
%
% The response is calculated as:
%
%                           a0 + a1*exp(s/fs) + ... + an*exp((n-1)s/fs)
%           H(s) = gain * -------------------------------------------
%                           b0 + b1*exp(s/fs) + ... + bm*exp((m-1)s/fs)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% The following call returns a cell array that contains the default parameter
% sets names:
%
% >> sets = resp(miir, 'Params')
%
% The following call returns a parameter list object that contains the
% default parameters for the jj-th set:
%
% >> pl = resp(miir, 'Params', sets{jj})
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = resp(miir, 'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = resp(miir, 'Category')
%
%
% VERSION: $Id: resp.m,v 1.10 2008/02/24 21:43:59 mauro Exp $
%
% NOTE:    Some of the code below is taken from Mathworks's treeplot.m
%
```

```
% HISTORY: 27-08-02 M Hewitson
%             Creation
%
%%%
```

```

% fill parameter list
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');
  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
  if size(f,1) > 1
    f = f.';
  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;
dresp = dresp .* filt.gain;  % apply the gain

% mag   = 20*log10(abs(dresp));
% phase = angle(dresp)*180/pi;


%% Create an analysis object

```

```
% create new output fsdata
fs = fsdata(f, dresp, filt.fs);
fs = set(fs, 'name', sprintf('resp(%s)', filt.name));

% create new output history
h = history(ALGONAME, VERSION, plo);

% make output analysis object
b = ao(fs, h);

% set name
b = setnh(b, 'name', sprintf('resp(%s)', filt.name));

% Outputs
if nargout == 0
  iplot(b)
end

if nargout == 1
  varargout{1} = b;
end
if nargout > 1
  error('incorrect output arguments');
end


%%%
```

```
0215
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo Exp $
%

```

```

v = z * 2;
idx = find(v>1);
v(idx) = 2-v(idx);


% Make window struct
w.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY overloads display functionality for xydata objects.

SYNOPSIS ^

function txt = display(xydatas, varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for xydata objects.

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

 DESCRIPTION: DISPLAY overloads display functionality for xydata objects.

 CALL:        txt    = display(xydata)

 INPUT:       xydata - tsdta object

 OUTPUT:      txt    - cell array with strings to display the xydata object

 VERSION:     $Id: display.m,v 1.7 2008/02/13 12:57:47 mauro Exp $

 HISTORY:     31-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.
- get GET gets xydata properties.

This function is called by:

- set SET sets a xydata property.

SOURCE CODE ^

```
function txt = display(xydatas, varargin)
% DISPLAY overloads display functionality for xydata objects.
%
%%%
```

```
0053   yinfo = whos('ydata');
0054
0055   txt{end+1} = sprintf('  name:  %s', get(xy, 'name'));
0056   txt{end+1} = sprintf('     x:  [%d %d], %s', xinfo.size(1), xinfo.size(2),
xinfo.class);
0057   txt{end+1} = sprintf('     y:  [%d %d], %s', yinfo.size(1), yinfo.size(2),
yinfo.class);
0058   txt{end+1} = sprintf('xunits:  %s', get(xy, 'xunits'));
0059   txt{end+1} = sprintf('yunits:  %s', get(xy, 'yunits'));
0060
0061   banner_end(1:length(banner)) = '-';
0062   txt{end+1} = banner_end;
0063
0064   txt{end+1} = ' ';
0065   txt{end+1} = ' ';
0066 end
0067
0068 if nargout == 0
0069   for ii=1:length(txt)
0070     disp(txt{ii});
0071   end
0072 end
0073
0074 % END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_hash

PURPOSE ^

LTPDA_HASH - Convert an input variable into a message digest using any of

SYNOPSIS ^

function h = ltpda_hash(inp,meth)

DESCRIPTION ^

```
 LTPDA_HASH - Convert an input variable into a message digest using any of
              several common hash algorithms

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

 LTPDA_HASH - Convert an input variable into a message digest using any of
              several common hash algorithms

 USAGE: h = hash(inp,'meth')

 inp  = input variable, of any of the following classes:
        char, uint8, logical, double, single, int8, uint8,
        int16, uint16, int32, uint32, int64, uint64
 h    = hash digest output, in hexadecimal notation
 meth = hash algorithm, which is one of the following:
        MD2, MD5, SHA-1, SHA-256, SHA-384, or SHA-512

 NOTES: (1) If the input is a string or uint8 variable, it is hashed
            as usual for a byte stream. Other classes are converted into
            their byte-stream values. In other words, the hash of the
            following will be identical:
                     'abc'
                     uint8('abc')
                     char([97 98 99])
            The hash of the follwing will be different from the above,
            because class "double" uses eight byte elements:
                     double('abc')
                     [97 98 99]
            You can avoid this issue by making sure that your inputs
            are strings or uint8 arrays.
        (2) The name of the hash algorithm may be specified in lowercase
```

```
            and/or without the hyphen, if desired. For example,
            h=hash('my text to hash','sha256');
        (3) Carefully tested, but no warranty. Use at your own risk.
        (4) Michael Kleder, Nov 2005
        (5) This is a direct copy of the code from Michael Kleder. Only
            the function name was changed to bring it in line with the
            LTPDA naming convention. (M Hewitson 15-09-07)

 EXAMPLE:

 algs={'MD2','MD5','SHA-1','SHA-256','SHA-384','SHA-512'};
 for n=1:6
     h=hash('my sample text',algs{n});
     disp([algs{n} ' (' num2str(length(h)*4) ' bits):'])
     disp(h)
 end

 $Id: ltpda_hash.m,v 1.1 2007/09/15 04:54:44 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpda_funchash LTPDA_FUNCHASH compute MD5 hash of an LTPDA function.

SOURCE CODE ^

```
function h = ltpda_hash(inp,meth)
% LTPDA_HASH - Convert an input variable into a message digest using any of
%              several common hash algorithms
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% LTPDA_HASH - Convert an input variable into a message digest using any of
%              several common hash algorithms
%
% USAGE: h = hash(inp,'meth')
%
% inp  = input variable, of any of the following classes:
%        char, uint8, logical, double, single, int8, uint8,
%        int16, uint16, int32, uint32, int64, uint64
% h    = hash digest output, in hexadecimal notation
% meth = hash algorithm, which is one of the following:
%        MD2, MD5, SHA-1, SHA-256, SHA-384, or SHA-512
%
```

```
% NOTES: (1) If the input is a string or uint8 variable, it is hashed
%            as usual for a byte stream. Other classes are converted into
%            their byte-stream values. In other words, the hash of the
%            following will be identical:
%                     'abc'
%                     uint8('abc')
%                     char([97 98 99])
%            The hash of the follwing will be different from the above,
%            because class "double" uses eight byte elements:
%                     double('abc')
%                     [97 98 99]
%            You can avoid this issue by making sure that your inputs
%            are strings or uint8 arrays.
%        (2) The name of the hash algorithm may be specified in lowercase
%            and/or without the hyphen, if desired. For example,
%            h=hash('my text to hash','sha256');
%        (3) Carefully tested, but no warranty. Use at your own risk.
%        (4) Michael Kleder, Nov 2005
%        (5) This is a direct copy of the code from Michael Kleder. Only
%            the function name was changed to bring it in line with the
%            LTPDA naming convention. (M Hewitson 15-09-07)
%
% EXAMPLE:
%
% algs={'MD2','MD5','SHA-1','SHA-256','SHA-384','SHA-512'};
% for n=1:6
%     h=hash('my sample text',algs{n});
%     disp([algs{n} ' (' num2str(length(h)*4) ' bits):'])
%     disp(h)
% end
%
% $Id: ltpda_hash.m,v 1.1 2007/09/15 04:54:44 hewitson Exp $
%
%

inp=inp(:);
% convert strings and logicals into uint8 format
if ischar(inp) || islogical(inp)
    inp=uint8(inp);
else % convert everything else into uint8 format without loss of data
    inp=typecast(inp,'uint8');
end

% verify hash method, with some syntactical forgiveness:
meth=upper(meth);
switch meth
    case 'SHA1'
        meth='SHA-1';
    case 'SHA256'
        meth='SHA-256';
```

```
    case 'SHA384'
        meth='SHA-384';
    case 'SHA512'
        meth='SHA-512';
    otherwise
end
algs={'MD2','MD5','SHA-1','SHA-256','SHA-384','SHA-512'};
if isempty(strmatch(meth,algs,'exact'))
    error(['Hash algorithm must be ' ...
        'MD2, MD5, SHA-1, SHA-256, SHA-384, or SHA-512']);
end

% create hash
x=java.security.MessageDigest.getInstance(meth);
x.update(inp);
h=typecast(x.digest,'uint8');
h=dec2hex(h)';
if(size(h,1))==1 % remote possibility: all hash bytes < 128, so pad:
    h=[repmat('0',[1 size(h,2)]);h];
end
h=lower(h(:)');
clear x
return
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY overloads display functionality for xyzdata objects.

SYNOPSIS ^

function txt = display(varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for xyzdata objects.

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

 DESCRIPTION: DISPLAY overloads display functionality for xyzdata objects.

 CALL:        txt    = display(xyzdata)

 INPUT:       xyzdata - an xyz data object

 OUTPUT:      txt    - cell array with strings to display the xyzdata object

 VERSION:     $Id: display.m,v 1.3 2008/02/13 13:22:45 mauro Exp $

 HISTORY:     24-12-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- get GET gets xyzdata properties.

This function is called by:

- set SET sets an xyzdata property.

SOURCE CODE ^

```
0001 function txt = display(varargin)
```

```
% DISPLAY overloads display functionality for xyzdata objects.
%
%%%
```

```
  zdata = get(xyz, 'z');
  xinfo = whos('xdata');
  yinfo = whos('ydata');
  zinfo = whos('zdata');

  txt{end+1} = sprintf('  name:  %s', get(xyz, 'name'));
  txt{end+1} = sprintf('     x:  [%d %d], %s', xinfo.size(1), xinfo.size(2), xinfo.class);
  txt{end+1} = sprintf('     y:  [%d %d], %s', yinfo.size(1), yinfo.size(2), yinfo.class);
  txt{end+1} = sprintf('     z:  [%d %d], %s', zinfo.size(1), zinfo.size(2), zinfo.class);
  txt{end+1} = sprintf('xunits:  %s', get(xyz, 'xunits'));
  txt{end+1} = sprintf('yunits:  %s', get(xyz, 'yunits'));
  txt{end+1} = sprintf('zunits:  %s', get(xyz, 'zunits'));

  banner_end(1:length(banner)) = '-';
  txt{end+1} = banner_end;

  txt{end+1} = ' ';
  txt{end+1} = ' ';
end

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_insert

PURPOSE ^

LTPDA_INSERT inserts values into a single row of a table using JDBC driver

SYNOPSIS ^

function message = ltpda_insert(conn, table, varargin)

DESCRIPTION ^

```
LTPDA_INSERT inserts values into a single row of a table using JDBC driver
specified by the input connection.

Usage: message = ltpda_insert(conn, table, 'field1', value1, 'field2',
                              value2,...)

M Hewitson

31-07-07

$Id: ltpda_insert.m,v 1.2 2008/02/22 09:25:18 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpda_obj_retrieve LTPDA_OBJ_RETRIEVE retrieves a collection of objects from an LTPDA repository.
- ltpda_obj_submit LTPDA_OBJ_SUBMIT submits the given collection of objects to an LTPDA Repository.

SOURCE CODE ^

```
function message = ltpda_insert(conn, table, varargin)

% LTPDA_INSERT inserts values into a single row of a table using JDBC driver
% specified by the input connection.
%
% Usage: message = ltpda_insert(conn, table, 'field1', value1, 'field2',
%                               value2,...)
%
% M Hewitson
```

```
%
% 31-07-07
%
% $Id: ltpda_insert.m,v 1.2 2008/02/22 09:25:18 hewitson Exp $
%

q = sprintf('INSERT INTO %s(',  table);
for j=1:2:length(varargin)
  q = [q varargin{j} ','];
end
q = q(1:end-1);

q = [q ') VALUES('];

for j=2:2:length(varargin)
  val = varargin{j};
  if ischar(val)
    q = [q '''' val '''' ','];
  elseif isnumeric(val)
    q = [q mat2str(val) ','];
  else
    error('unknown data type');
  end

end

q = q(1:end-1);
q = [q ');'];

try
  curs = exec(conn, q);
  message = curs.Message;
  close(curs);
catch
  error('### ltpda_insert: failed to execute query: %s\nServer returned %s', q, curs.Message);
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

resp

PURPOSE ^

RESP returns the complex response of the pole object.

SYNOPSIS ^

function [f,r] = resp(varargin)

DESCRIPTION ^

```
 RESP returns the complex response of the pole object.

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

 DESCRIPTION: RESP returns the complex response of the pole object.

 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, pl);         % compute response from parameter list.
              [f,r] = resp(p);             % compute response

 REMARK:      This is just a helper function. This function should only be
              called from class functions.

 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

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

 The following call returns a cell array that contains the default parameter
 sets names:

 >> sets = resp(pole, 'Params')

 The following call returns a parameter list object that contains the
 default parameters for the jj-th set:
```

```
>> pl = resp(pole, 'Params', sets{jj})

The following call returns a string that contains the routine CVS version:

>> version = resp(pole, 'Version')

The following call returns a string that contains the routine category:

>> category = resp(pole, 'Category')


VERSION:     $Id: resp.m,v 1.9 2008/02/24 21:43:59 mauro Exp $

HISTORY:     03-04-2007 Hewitson
                Creation

%%%
```

```
% REMARK:      This is just a helper function. This function should only be
%              called from class functions.
%
% 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
%
%%%
```

```
    else
      f = getDefaultPL(varargin{3});
    end
    return
  elseif strcmp(in, 'Version')
    f = VERSION;
    return
  elseif strcmp(in, 'Category')
    f = CATEGORY;
    return
  end
end

%% Input objects checks
if nargin < 1
  error('### incorrect number of inputs.')
end

% look at inputs
p = varargin{1};
if ~isa(p, 'pole')
  error('### first argument should be a pole object.');
end


% Now look at the pole
f0 = get(p, 'f');
Q  = get(p, 'q');



%% check inputs

if nargin == 1
  %% a = resp(p); %%
  pl =  getDefaultPL(p);

elseif nargin == 2
  if isa(varargin{2}, 'plist')
    %% a = resp(p, plist); %%
    pl = varargin{2};
    if ~isempty(find(pl, 'f'))
      pl = combine(pl, getDefaultPL('List'));
    else
      pl = combine(pl, getDefaultPL('Range'));
    end
  else
    %% a = resp(p, f); %%
    f = varargin{2};
    pl = plist('f',f);
  end
```

```
elseif nargin == 4
  %% a = resp(p, f1, f2, nf); %%
  f1 = varargin{2};
  f2 = varargin{3};
  nf = varargin{4};

  pl = plist('f1',f1,...
    'f2',f2,...
    'nf',nf, ...
    'scale', 'lin');

else
  error('### incorrect number of inputs.');
end


%% Compute or extract the frequency vector
if ~isempty(find(pl, 'f'))
  % Compute from frequency list
  f     = find(pl, 'f');
  % Want to deal with rows
  if size(f,1) > 1
    f = f.';
  end
else
  % Compute from frequency range
  f1 = find(pl, 'f1');
  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


%% 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


```

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

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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo Exp $
%

```

```
z = z * 2 * pi;
v = 0.2810639 - 0.5208972 * cos (z) + 0.1980399 * cos (2 * z);

% Make window struct
w.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

display

PURPOSE ^

DISPLAY overloads display functionality for zero objects.

SYNOPSIS ^

function txt = display(p, varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for zero objects.

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

 DESCRIPTION: DISPLAY overloads display functionality for zero objects.

 CALL:        txt  = display(zero)

 INPUT:       zero - zero object

 OUTPUT:      txt  - cell array with strings to display the zero object

 VERSION:     $Id: display.m,v 1.5 2008/02/12 19:54:16 hewitson Exp $

 HISTORY:     30-01-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a zero object into a string.
- get GET get zero properties.

This function is called by:

- set SET set a zero property.

SOURCE CODE ^

```
function txt = display(p, varargin)
% DISPLAY overloads display functionality for zero objects.
%
%%%
```

```

  % display
  if ~isempty(f)
    if q > 0.5
      txt{end+1} = [ sprintf('%s: %g Hz, Q=%g [', name, f, q) num2str(ri(1))
']'];
    else
      txt{end+1} = sprintf('%s: %g Hz', name, f);
    end
  end

  banner_end(1:length(banner)) = '-';
  txt{end+1} = banner_end;

end

if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
end


```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_io_collect

PURPOSE ^

LTPDA_IO_COLLECT collects all AOs and plists from the inputs.

SYNOPSIS ^

function varargout = ltpda_io_collect(varargin)

DESCRIPTION ^

```
 LTPDA_IO_COLLECT collects all AOs and plists from the inputs.
%%%
```

SOURCE CODE ^

```
function varargout = ltpda_io_collect(varargin)

% LTPDA_IO_COLLECT collects all AOs and plists from the inputs.
%%%
```

```


%-----------------------------------------------------------------------
% Unpack AOs from input
function aos = unpack_aos(ain)

aos = {};
for jj=1:numel(ain)
  a = ain(jj);
  if numel(a) > 1
    aos = [aos unpack_aos(a)];
  else
    aos = [aos {a}];
  end
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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

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

 The following call returns a cell array that contains the default parameter
 sets names:

 >> sets = resp(pzmodel, 'Params')

 The following call returns a parameter list object that contains the
 default parameters for the jj-th set:

 >> pl = resp(pzmodel, 'Params', sets{jj})
```

```
The following call returns a string that contains the routine CVS version:

>> version = resp(pzmodel, 'Version')

The following call returns a string that contains the routine category:

>> category = resp(pzmodel, 'Category')


VERSION:     $Id: resp.m,v 1.16 2008/02/24 21:43:59 mauro Exp $

HISTORY:     03-04-2007 M Hewitson
                Creation

%%%
```

```
%              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
%
%%%
```

```
    if nargin == 2
      varargout{1} = getDefaultPL();
    else
      varargout{1} = getDefaultPL(varargin{3});
    end
    return
  elseif strcmp(in, 'Version')
    varargout{1} = VERSION;
    return
  elseif strcmp(in, 'Category')
    varargout{1} = CATEGORY;
    return
  end
end

%% Input objects checks
if nargin < 1
  error('### incorrect number of inputs.')
end

% look at inputs
pzm = varargin{1};
if ~isa(pzm, 'pzmodel')
  error('### first argument should be a pzmodel object.');
end


% Now look at the model
name  = get(pzm, 'name');
gain  = get(pzm, 'gain');
poles = get(pzm, 'poles');
zeros = get(pzm, 'zeros');
np    = length(poles);
nz    = length(zeros);

%% check inputs

if nargin == 1
  %% a = resp(pzm); %%
  pl =  getDefaultPL(pzm);

elseif nargin == 2
  if isa(varargin{2}, 'plist')
    %% a = resp(pzm, plist); %%
    pl = varargin{2};
    if ~isempty(find(pl, 'f'))
      pl = combine(pl, getDefaultPL('List'));
    else
      pl = combine(pl, getDefaultPL('Range'));
    end
  else
    %% a = resp(pzm, f); %%
```

```
    f = varargin{2};
    pl = plist('f',f);
  end
elseif nargin == 4
  %% a = resp(pzm, f1, f2, nf); %%
  f1 = varargin{2};
  f2 = varargin{3};
  nf = varargin{4};

  pl = plist('f1',f1,...
    'f2',f2,...
    'nf',nf, ...
    'scale', 'lin');

else
  error('### incorrect number of inputs.');
end


%% Compute or extract the frequency vector
if ~isempty(find(pl, 'f'))
  % Compute from frequency list
  f     = find(pl, 'f');
  % Want to deal with rows
  if size(f,1) > 1
    f = f.';
  end
else
  % Compute from frequency range
  f1 = find(pl, 'f1');
  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

%% make output plist
plo = combine(plist('pzmodel', pzm), pl);


%% Now compute the response
r = gain;
for j=1:np
  if ~isnan(poles(j).f)
    [f, pr] = resp(poles(j),f);
    r = r .* pr;
  end
```

```
end
for j=1:nz
  if ~isnan(zeros(j).f)
    [f, zr] = resp(zeros(j),f);
    r = r .* zr;
  end
end

%% ---------- Build output ao

% create new output fsdata
fs = fsdata(f, r, 1);
fs = set(fs, 'name', sprintf('resp(%s)', name));

% create new output history
h = history(ALGONAME, VERSION, plo);

% make output analysis object
b = ao(fs, h);

% set name
b = setnh(b, 'name', sprintf('resp(%s)', name), ...
  'mfilename', ALGONAME);

% Outputs
if nargout == 0
  iplot(b)
elseif nargout == 1
  varargout{1} = b;
else
  error('### incorrect output arguments');
end


%% %%%%
```

```
end

set = varargin{1};

if isa(set, 'pzmodel')
  f1 = getlowerFreq(set)/10;
  f2 = getupperFreq(set)*10;
  nf = 10000;
  scale = 'lin';
  plo = plist('f1', f1,...
    'f2', f2,...
    'nf', nf,...
    'scale', scale);
else
  switch set
    case 'List'
      plo = plist('f', []);
    case 'Range'
      plo = plist('f1', 0.001,...
        'f2', 1,...
        'nf', 10000,...
        'scale', 'lin');
    otherwise
      plo = plist();
  end
end

% END


```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_isobject

PURPOSE ^

LTPDA_ISOBJECT checks that the input objects are one of the LTPDA object

SYNOPSIS ^

function varargout = ltpda_isobject(varargin)

DESCRIPTION ^

```
 LTPDA_ISOBJECT checks that the input objects are one of the LTPDA object
 types.

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

 DESCRIPTION: LTPDA_ISOBJECT checks that the input objects are one of the LTPDA
object
              types.

 CALL:        result  = ltpda_isobject(a1)
              classes = ltpda_isobject()

 INPUTS:      objects

 OUTPUTS:     result == 1 if all input objects are LTPDA objects
              result == 0 otherwise
              classes - a list of recognised LTPDA object types

 VERSION:     $Id: ltpda_isobject.m,v 1.3 2008/02/14 16:04:30 mauro Exp $

 HISTORY: 09-05-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ltpda_classes LTPDA_CLASSES lists all the LTPDA object types.

This function is called by:

SUBFUNCTIONS ^

- function varargout = isltpdaobj(obj, classes)

SOURCE CODE ^

```
function varargout = ltpda_isobject(varargin)

% LTPDA_ISOBJECT checks that the input objects are one of the LTPDA object
% types.
%
%%%
```

```
        end
      else
        for k=1:length(obj)
          % check each one

          % Stupid hack required because database objects can't cope with
          % begin indexed like conn(1) - all other objects work for this
          % call.
          if k==1
            if ~isltpdaobj(obj, classes)
              varargout{1} = 0;
              return
            end
          else
            if ~isltpdaobj(obj(k), classes)
              varargout{1} = 0;
              return;
            end
          end
        end
      end
    end
  else
    varargout{1} = 0;
    return;
  end
end

varargout{1} = 1;

%-----------------------------------------------------------------------
% Test a single object
%
function varargout = isltpdaobj(obj, classes)

found = 0;
for k=1:length(classes)
  type = classes{k};
  if isa(obj, type)
    found = 1;
  end
end
if ~found
  varargout{1} = 0;
  return
else
  varargout{1} = 1;
end

```

resp

PURPOSE ^

RESP returns the complex response of the zero object.

SYNOPSIS ^

function [f,r] = resp(varargin)

DESCRIPTION ^

```
 RESP returns the complex response of the zero object.

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

 DESCRIPTION: RESP returns the complex response of the zero object.

 CALL:        [f,r] = resp(z, f);          % compute response for vector f
              [f,r] = resp(z, f1, f2, nf); % compute response from f1 to f2
                                           % in nf steps.
              [f,r] = resp(z, pl);         % compute response from parameter list.
              [f,r] = resp(z);             % compute response

 REMARK:      This is just a helper function. This function should only be
              called from class functions.

 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

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

 The following call returns a cell array that contains the default parameter
 sets names:

 >> sets = resp(zero, 'Params')

 The following call returns a parameter list object that contains the
 default parameters for the jj-th set:
```

```
>> pl = resp(zero, 'Params', sets{jj})

The following call returns a string that contains the routine CVS version:

>> version = resp(zero, 'Version')

The following call returns a string that contains the routine category:

>> category = resp(zero, 'Category')


VERSION:     $Id: resp.m,v 1.9 2008/02/24 21:43:59 mauro Exp $

HISTORY:     03-04-2007 Hewitson
                Creation

%%%
```

```
% REMARK:      This is just a helper function. This function should only be
%              called from class functions.
%
% 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
%
%%%
```

```
    else
      f = getDefaultPL(varargin{3});
    end
    return
  elseif strcmp(in, 'Version')
    f = VERSION;
    return
  elseif strcmp(in, 'Category')
    f = CATEGORY;
    return
  end
end

%% Input objects checks
if nargin < 1
  error('### incorrect number of inputs.')
end

% look at inputs
z = varargin{1};
if ~isa(z, 'zero')
  error('### first argument should be a zero object.');
end


% Now look at the zero
f0 = get(z, 'f');
Q  = get(z, 'q');



%% check inputs

if nargin == 1
  %% a = resp(z); %%
  pl =  getDefaultPL(z);

elseif nargin == 2
  if isa(varargin{2}, 'plist')
    %% a = resp(z, plist); %%
    pl = varargin{2};
    if ~isempty(find(pl, 'f'))
      pl = combine(pl, getDefaultPL('List'));
    else
      pl = combine(pl, getDefaultPL('Range'));
    end
  else
    %% a = resp(z, f); %%
    f = varargin{2};
    pl = plist('f',f);
  end
```

```
elseif nargin == 4
  %% a = resp(z, f1, f2, nf); %%
  f1 = varargin{2};
  f2 = varargin{3};
  nf = varargin{4};

  pl = plist('f1',f1,...
    'f2',f2,...
    'nf',nf, ...
    'scale', 'lin');

else
  error('### incorrect number of inputs.');
end


%% Compute or extract the frequency vector
if ~isempty(find(pl, 'f'))
  % Compute from frequency list
  f     = find(pl, 'f');
  % Want to deal with rows
  if size(f,1) > 1
    f = f.';
  end
else
  % Compute from frequency range
  f1 = find(pl, 'f1');
  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


%% Now compute the response
if Q>=0.5
  re = 1 - (f.^2./f0^2);
  im = f ./ (f0*Q);
  r = complex(re, im);
else
  re = 1;
  im = f./f0;
  r = complex(re, im);
end


```

```

%% %%%%
```

win_HFT116D

PURPOSE ^

WIN_HFT116D return HFT116D window sample for input z.

SYNOPSIS ^

function w = win_HFT116D(w, z)

DESCRIPTION ^

```
WIN_HFT116D return HFT116D window sample for input z.

M Hewitson 18-05-07

$Id: win_HFT116D.m,v 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_isuserobject

PURPOSE ^

LTPDA_ISUSEROBJECT checks that the input objects are one of the LTPDA user object

SYNOPSIS ^

function varargout = ltpda_isuserobject(varargin)

DESCRIPTION ^

```
 LTPDA_ISUSEROBJECT checks that the input objects are one of the LTPDA user object
 types.

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

 DESCRIPTION: LTPDA_ISUSEROBJECT checks that the input objects are one of the
              LTPDA user object types.

 CALL:        result  = ltpda_isuserobject(a1)
              classes = ltpda_isuserobject()

 INPUTS:      objects

 OUTPUTS:     result == 1 if all input objects are LTPDA objects
              result == 0 otherwise
              classes - a list of recognised LTPDA object types

 VERSION:     $Id: ltpda_isuserobject.m,v 1.5 2008/02/14 16:04:30 mauro Exp $

 HISTORY: 09-05-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ltpda_user_classes ltpda_user_classes lists all the LTPDA user object types.

This function is called by:

SUBFUNCTIONS ^

- function varargout = isltpdaobj(obj, classes)

SOURCE CODE ^

```
function varargout = ltpda_isuserobject(varargin)

% LTPDA_ISUSEROBJECT checks that the input objects are one of the LTPDA user object
% types.
%
%%%
```

```
        end
      else
        for k=1:length(obj)
          % check each one

          % Stupid hack required because database objects can't cope with
          % begin indexed like conn(1) - all other objects work for this
          % call.
          if k==1
            if ~isltpdaobj(obj, classes)
              varargout{1} = 0;
              return
            end
          else
            if ~isltpdaobj(obj(k), classes)
              varargout{1} = 0;
              return;
            end
          end
        end
      end
    end
  else
    varargout{1} = 0;
    return;
  end
end

varargout{1} = 1;

%-----------------------------------------------------------------------
% Test a single object
%
function varargout = isltpdaobj(obj, classes)

found = 0;
for k=1:length(classes)
  type = classes{k};
  if isa(obj, type)
    found = 1;
  end
end
if ~found
  varargout{1} = 0;
  return
else
  varargout{1} = 1;
end

```

retrieve

PURPOSE ^

RETRIEVE analysis objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE analysis objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE analysis objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.9 2007/08/31 17:40:08 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE analysis objects with given ids from the LTPDA repository specified
by the input
```

```
% database connection.
%
%%%
```

```
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.3 2008/02/17 18:46:31 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.3 2008/02/17 18:46:31 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_jacobian

PURPOSE ^

subfunction to handle exceptions when calling the jacobian function

SYNOPSIS ^

function dM = ltpda_jacobian(M,var, order)

DESCRIPTION ^

```
 subfunction to handle exceptions when calling the jacobian function
%%%
```

- ltpda_ss_modify paramSubstitute enables to modify parameters and retrieve numerical

SOURCE CODE ^

```
function dM = ltpda_jacobian(M,var, order)
% subfunction to handle exceptions when calling the jacobian function
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

retrieve

PURPOSE ^

RETRIEVE cdata objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE cdata objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE cdata objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE cdata objects with given ids from the LTPDA repository specified by
the input
```

```
% database connection.
%
%%%
```

```
0056
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for cdata objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for cdata objects.

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

 DESCRIPTION: EQ overloads the == operator for cdata objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input cdata objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two cdata objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.15 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

retrieve

PURPOSE ^

RETRIEVE fsdata objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE fsdata objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE fsdata objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE fsdata objects with given ids from the LTPDA repository specified by
the input
```

```
% database connection.
%
%%%
```

```
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for fsdata objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for fsdata objects.

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

 DESCRIPTION: EQ overloads the == operator for fsdata objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input fsdata objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two fsdata objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.9 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_kalman_filter

PURPOSE ^

Kalman filter

SYNOPSIS ^

function ao_out = ltpda_ss_kalman_filter(ao_Y, ao_in_list, data_plist, syst)

DESCRIPTION ^

```
Kalman filter
%%%
```

```
% OUTPUTS: ao_out _ output analysis object
%
% ***** THERE ARE NO DEFAULT PARAMETERS *****
%
% &&VERSION: $Id: ltpda_kalman_filter.m,v 1.2 2008/02/25 12:43:44 adrien Exp $
%
% HISTORY: 19-02-2008 A Grynagier
% 17-02-2008 A Grynagier
% Creation 30-01-2008 M Weyrich
%
% TO DO: options to be defined
% implement input type of three kinds:
% - AO
% - Constant
% - gaussian white noise
% use LTPDA functions to generate white noise
%%%
```

```
% input covariances
Qs = find(data_plist, 'Qs');
Rs = find(data_plist, 'Rs');
Noise_Names = find(data_plist, 'Noise_Names');
Q = zeros(nss);
for i=1:length(Noise_Names)
    for j=1:NBINPUTS
        if isequal(Noise_Names{i},INPUTNAMES{j} )
            Q = Q + B{j}*Qs{i}*B{j}'; %assumption in this covariance addition is that the inputs are not correlated
            R = R + D{j}*Rs{i}*D{j}';
        end
    end
end

%% read out AO (time-series data)
ao_in = cell(NBINPUTS,1);
for i=1:NBINPUTS
    if INPUTISUSED(i)
        ao_in{i} = find(ao_in_list, INPUTNAMES{i});
        if isequal(class(ao_in{i}), 'ao')
            if not(isequal(ao_in{i}.data.x,{})) %making sure ao is not empty
                t = ao_in{i}.data.x;
                dt = diff(t);
                if sum(not(isequal(dt,timestep)))>1
                    msg = ['error in input number ', i , ' called ', INPUTNAMES{i}, 'because wrong timestep for simulation'];
                    error(msg);
                end
                if not( isequal( size(B{i},1), size(D{i},1), size(ao_in{i}.data.y,1) ) )
                    msg = ['error in input number ', i , ' called ', INPUTNAMES{i}, 'because wrong input size'];
                    error(msg);
                end
            end
        end
    end
end
%% getting correct number of samples
nsamples = size(ao_Y{i}.data.y,2);




%% Kalman filter vector allocation
P0=eye(nss);
X_est=zeros(size(A,1),nsamples+1);
X_upd=zeros(size(A,1),nsamples+1);
X_upd(:,1)=X(:,1);
Y_tilde=zeros(size(Y,1),nsamples+1);

```

```
%%%%%%%%%%%%%%%%%%%
%%% Kalman filter %% chg
%%%%%%%%%%%%%%%%%%%

for i=2:sim_n %chg
    IncrementX = zeros(nss,1);
    IncrementY = zeros(size(C,1),1);
    for j=1:length(ao_in)
        IncrementX = IncrementX +  B{j}*ao_in{j}.data.y(:,i) ;
        IncrementY = IncrementY +  D{j}*ao_in{j}.data.y(:,i) ;
    end
    X_est(:,i) = A*X_upd(:,i-1)+IncrementX; %Estimate
    Y_tilde(:,i) = ao_Y.data.y(:,i)-C*X_est(:,i)-IncrementY; %Measurement residual chg dEst forgotten
    P = A*P0*A'+Q;
    S = C*P*C'+R;  %Covariance of Y-tilde
    K = P*C'*inv(S); %K. gain
    X_upd(:,i) = X_est(:,i)+K*Y_tilde(:,i); %Update
    P0 = (eye(size(K,1))-K*C)*P;
end


%% data object
if exist('t','var')
    ts = tsdata(t,[X_est; Ytilde]) ;
else
    ts = tsdata(1:nsamples,[X_est; Ytilde]) ;
end

%% history object, old history from ao_in plus new
hi_old = ao_Y.hist ;
hi_new = history('Kalman','version', syst);
hi = [hi_old hi_new];
h = history(hi);

%% construct output analysis object
ao_out  = ao(ts,h) ;
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

retrieve

PURPOSE ^

RETRIEVE history objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE history objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE history objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE history objects with given ids from the LTPDA repository specified
by the input
```

```
% database connection.
%
%%%
```

```
0056
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for history objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for history objects.

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

 DESCRIPTION: EQ overloads the == operator for history objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input history objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two history objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.6 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_label

PURPOSE ^

LTPDA_LABEL makes the input string into a suitable string for using on plots.

SYNOPSIS ^

function s = ltpda_label(si)

DESCRIPTION ^

```
 LTPDA_LABEL makes the input string into a suitable string for using on plots.

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

 DESCRIPTION: LTPDA_LABEL makes the input string into a suitable string for
              using on plots.

 CALL:        s = ltpda_label(sin)

 INPUTS:      sin - input string

 OUTPUTS:     s   - suitable string for the plot functions

 VERSION: $Id: ltpda_label.m,v 1.2 2007/07/13 12:17:39 ingo Exp $

 HISTORY: 11-02-2007 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function s = ltpda_label(si)
% LTPDA_LABEL makes the input string into a suitable string for using on plots.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%
% DESCRIPTION: LTPDA_LABEL makes the input string into a suitable string for
%              using on plots.
%
% CALL:       s = ltpda_label(sin)
%
% INPUTS:     sin - input string
%
% OUTPUTS:    s   - suitable string for the plot functions
%
% VERSION: $Id: ltpda_label.m,v 1.2 2007/07/13 12:17:39 ingo Exp $
%
% HISTORY: 11-02-2007 M Hewitson
%             Creation
%
%%%
```

retrieve

PURPOSE ^

RETRIEVE mfir objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE mfir objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE mfir objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:07 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE mfir objects with given ids from the LTPDA repository specified by
the input
```

```
% database connection.
%
%%%
```

```
0056
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.3 2008/02/17 18:46:31 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.3 2008/02/17 18:46:31 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for mfir objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for mfir objects.

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

 DESCRIPTION: EQ overloads the == operator for mfir objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input mfir objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two mfir objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.5 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 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 1.10 2008/02/21 08:52:17 hewitson Exp $

%%%
```

```
%   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 1.10 2008/02/21 08:52:17 hewitson Exp $
%
%%%
```

```
[as, ps, invars] = collect_inputs(varargin, in_names);

% initialise output array
bs = [];

% check plist
if isempty(ps)
  pl = getDefaultPL();
else
  pl = combine(ps, getDefaultPL);
end

% Go through each input AO and build a matrix of signals
na    = length(as);

%---------------- Resample all AOs

disp('*** Resampling AOs...');
fsmax = findFsMax(as);
fspl  = plist(param('fsout', fsmax));
bs = [];
for i=1:na
  a = as(i);
  % check this is a time-series object
  if ~isa(a.data, 'tsdata')
    error('### ltpda_lcohere requires tsdata (time-series) inputs.');
  end
  % Check Fs
  if a.data.fs ~= fsmax
    warning(sprintf('!!! Resampling AO %s/%s to %f Hz', a.name, a.data.name, fsmax));
    a = resample(a, fspl);
  end
  bs = [bs a];
end
as = bs;
na = length(as);
clear bs;
disp('*** Done.');

%---------------- Truncate all vectors

% Get shortest vector
disp('*** Truncating all vectors...');
lmin = findShortestVector(as);
nsecs = lmin / fsmax;
bs = [];
for i=1:na
  a = as(i);
  if len(a) ~= lmin
    warning(sprintf('!!! Truncating AO %s/%s to %d secs', a.name, a.data.name, nsecs));
```

```
    bs = [bs  select(a, 1:lmin)];
  else
    bs = [bs a];
  end
end
as = bs;
clear bs;
disp('*** Done.');


%---------------- Build signal Matrix
N     = len(as(1)); % length of first signal
iS    = zeros(na, N);
for j=1:na
  a = as(j);
  iS(j,:) = a.data.y;
end

%---------------- check input parameters

% Averaging
Kdes = find(pl, 'Kdes');   % desired averages
Kmin = find(pl, 'Kmin');   % minimum averages
Jdes = find(pl, 'Jdes');   % num desired spectral frequencies
Win   = find(pl, 'Win');   % Window object
Nolap = find(pl, 'Nolap');    % desired overlap
if Nolap == -1
  % use window function rov
  Nolap = Win.rov/100;
  disp(sprintf('! Using recommended overlap of %d', Nolap))
  pl = pset(pl, 'Nolap', Nolap);
end
Order = find(pl, 'Order');    % desired overlap
Lmin = find(pl, 'Lmin'); % Minimum segment length
if isempty(Lmin)
  Lmin = 0;
end

% Debug
debug = find(pl, 'DEBUG');

%---------------- Get frequency vector
[f, r, m, L, K] = ltpda_ltf_plan(lmin, fsmax, Nolap, 1, Lmin, Jdes, Kdes);
% [f,r,m,L] = ltpda_compute_f(fsmax, lmin, Kdes, Kmin, Jdes, Nolap, debug);

%---------------- compute CPSD Estimates

Txy = mlcohere(iS, f, r, m, L, fsmax, Win, Order, Nolap*100, Lmin);


%   if nargout == 2
%     disp('** Computing Coherence matrix with variance estimates')
```

```
%     tic
%     [Txy, ENBW, Sig] = mlcohere(iS, f, r, m, L, fsmax, Win, Order);
%     toc
%   else
%     disp('** Computing Coherence matrix')
%     tic
%     [Txy, ENBW]  = mlcohere(iS, f, r, m, L, fsmax, Win, Order);
%     toc
%   end

%---------------- Build output Matrix of AOs
disp('** Building output AO matrix')
bs  = cell(na);
for j=1:na  % Loop over input signal
  bi = as(j);
  for k=1:na  % Loop over output signal
    bo = as(k);
    % create new output fsdata
    fs = fsdata(f, squeeze(Txy(j,k,:)), fsmax);
    fs = set(fs, 'name', sprintf('LCOHERE(%s->%s)', bi.data.name, bo.data.name));
    fs = set(fs, 'yunits', [bo.data.yunits '/' bi.data.yunits]);
    % create new output history
    h = history(ALGONAME, VERSION, pl, [bi.hist bo.hist]);
    if na == length(invars)
      h = set(h, 'invars', invars([j k]));
      nk = invars{k};
      nj = invars{j};
    else
      nk = bo.name;
      nj = bi.name;
    end
    % make output analysis object
    b = ao(fs, h);
    % set name
    b = setnh(b, 'name', sprintf('LCOHERE(%s->%s)', nj, nk));
    % add to outputs
    bs(j,k) = {b};
  end % End loop over output signal
end % End loop over input signal

% now we have a cell matrix of AOs but
% we want a normal matrix
b  = [bs{:}];
bs = reshape(b, na, na);
varargout{1} = bs;

%-----------------------------------------------------------------------
% 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


%-----------------------------------------------------------------------
% 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

%-----------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

disp('* creating default plist...');
plo = plist();
plo = append(plo, param('Kdes', 100));
plo = append(plo, param('Jdes', 1000));
plo = append(plo, param('Kmin', 2));
plo = append(plo, param('Lmin', 0));
plo = append(plo, param('Win',  specwin('Kaiser', 1, 200)));
plo = append(plo, param('Nolap', -1));
plo = append(plo, param('Order', 0));
disp('* done.');


%-----------------------------------------------------------------------
% Compute all TF combinations
function varargout = mlcohere(varargin)

% MLCOHERE
%
% Txy = mlcohere(X,f,r,m,L,fs,win,order,olap,Lmin)
%
% M Hewitson 19-05-07
%
%

% Get inputs
X     = varargin{1};
f     = varargin{2};
r     = varargin{3};
m     = varargin{4};
```

```
L     = varargin{5};
fs    = varargin{6};
win   = varargin{7};
order = varargin{8};
olap  = varargin{9};
Lmin  = varargin{10};


% --- Prepare some variables
twopi      = 2.0*pi;
si         = size(X);
nc         = si(1);
nf         = length(f);
Txy        = zeros(nc,nc,nf);
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.name
      case 'Kaiser'
        win = specwin(win.name, l, win.psll);
      otherwise
        win = specwin(win.name, 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
    disp(sprintf('++ 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
      if k>j
        % Core cross-DFT part in C-mex file
        xi    = X(j,:);
```

```
        xo    = X(k,:);
        % We need cross-spectrum and Power spectrum
        [XY, XX, YY, nsegs] = ltpda_dft(xi, xo, 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 loop over output channels
  end % End loop over input channels
end % End loop over frequencies

% Fill other elements
for j=1:nc
  for k=1:nc
    if k<j
      Txy(j,k,:) = Txy(k,j,:);
    end
  end
end

% Set output
varargout{1} = Txy;


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

retrieve

PURPOSE ^

RETRIEVE miir objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE miir objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE miir objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE miir objects with given ids from the LTPDA repository specified by
the input
```

```
% database connection.
%
%%%
```

```
0056
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function w = win_HFT70(w, z)
0002
0003 % WIN_HFT70 return HFT70 window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT70.m,v 1.2 2008/02/13 18:38:02 ingo Exp $
0008 %
0009
```

```
z = z * 2 * pi;
v =  (1 - 1.90796 * cos (z) + ...
      1.07349 * cos (2 * z) - 0.18199 * cos (3 * z));

% Make window struct
w.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for miir objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for miir objects.

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

 DESCRIPTION: EQ overloads the == operator for miir objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input miir objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two miir objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.5 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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 1.12 2008/02/21 08:52:17 hewitson Exp $

%%%
```

```
%     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 1.12 2008/02/21 08:52:17 hewitson Exp $
%
%%%
```

```
  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, ps, invars] = collect_inputs(varargin, in_names);

% initialise output array
bs = [];

% check plist
if isempty(ps)
  pl = getDefaultPL();
else
  pl = combine(ps, getDefaultPL);
end

% Go through each input AO and build a matrix of signals
na    = length(as);

%---------------- Resample all AOs

disp('*** Resampling AOs...');
fsmax = findFsMax(as);
fspl  = plist(param('fsout', fsmax));
bs = [];
for i=1:na
  a = as(i);
  % check this is a time-series object
  if ~isa(a.data, 'tsdata')
    error('### ltpda_lcpsd requires tsdata (time-series) inputs.');
  end
  % Check Fs
  if a.data.fs ~= fsmax
    warning(sprintf('!!! Resampling AO %s/%s to %f Hz', a.name, a.data.name, fsmax));
    a = resample(a, fspl);
  end
  bs = [bs a];
end
as = bs;
clear bs;
disp('*** Done.');

%---------------- Truncate all vectors

```

```
% Get shortest vector
disp('*** Truncating all vectors...');
lmin = findShortestVector(as);
nsecs = lmin / fsmax;
bs = [];
for i=1:na
  a = as(i);
  if len(a) ~= lmin
    warning(sprintf('!!! Truncating AO %s/%s to %d secs', a.name, a.data.name, nsecs));
    bs = [bs  select(a, 1:lmin)];
  else
    bs = [bs a];
  end
end
as = bs;
na = length(as);
clear bs;
disp('*** Done.');


%---------------- Build signal Matrix
N     = len(as(1)); % length of first signal
iS    = zeros(na, N);
for j=1:na
  a = as(j);
  iS(j,:) = a.data.y;
end

%---------------- check input parameters

% Averaging
Kdes = find(pl, 'Kdes');   % desired averages
Kmin = find(pl, 'Kmin');    % minimum averages
Jdes = find(pl, 'Jdes');    % num desired spectral frequencies
if Jdes == -1
  Jdes = round(fsmax/4);
  disp(sprintf('! Using default Jdes of %d', Jdes))
  pl = pset(pl, 'Jdes', Jdes);
end
Win   = find(pl, 'Win');   % Window object
Nolap = find(pl, 'Nolap');    % desired overlap
if Nolap == -1
  % use window function rov
  Nolap = Win.rov/100;
  disp(sprintf('! Using recommended overlap of %d', Nolap))
  pl = pset(pl, 'Nolap', Nolap);
end
Order = find(pl, 'Order');    % desired overlap
if isempty(Order)
  Order = 0;
  disp(sprintf('! Using default detrending order of %d', Order))
```

```
  pl = append(pl, param('Order', Order));
end
Lmin = find(pl, 'Lmin'); % Minimum segment length
if isempty(Lmin)
  Lmin = 0;
end

%----------------- Get frequency vector
[f, r, m, L, K] = ltpda_ltf_plan(lmin, fsmax, Nolap, 1, Lmin, Jdes, Kdes);

%----------------- compute CPSD Estimates
Txy = mlcpsd(iS, f, r, m, L, fsmax, Win, Order, Nolap*100, Lmin);

%----------------- Build output Matrix of AOs
disp('** Building output AO matrix')
bs  = cell(na);
sig = cell(na);
for j=1:na  % Loop over input signal
  bi = as(j);
  for k=1:na  % Loop over output signal
    bo = as(k);
    % create new output fsdata
    fs = fsdata(f, squeeze(Txy(j,k,:)), fsmax);
    fs = set(fs, 'name', sprintf('LCPSD(%s->%s)', bi.data.name, bo.data.name));
    fs = set(fs, 'yunits', [bo.data.yunits '/' bi.data.yunits]);
    % create new output history
    h = history(ALGONAME, VERSION, pl, [bi.hist bo.hist]);
    h = set(h, 'invars', invars([j k]));
    % make output analysis object
    b = ao(fs, h);
    % set name
    if isempty(invars{k})
      nk = bo.name;
    else
      nk = invars{k};
    end
    if isempty(invars{j})
      nj = bi.name;
    else
      nj = invars{j};
    end
    b = setnh(b, 'name', sprintf('LCPSD(%s->%s)', nj, nk));
    % add to outputs
    bs(j,k) = {b};
  end % End loop over output signal
end % End loop over input signal

% now we have a cell matrix of AOs but
% we want a normal matrix
b  = [bs{:}];
bs = reshape(b, na, na);
varargout{1} = bs;
```

```
if nargout == 2
  b  = [sig{:}];
  s  = reshape(b, na, na);
  varargout{2} = s;
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


%-------------------------------------------------------------------------
% 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

%-------------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('Kdes', 100));
plo = append(plo, param('Jdes', 1000));
plo = append(plo, param('Kmin', 2));
plo = append(plo, param('Lmin', 0));
plo = append(plo, param('Win',  specwin('Kaiser', 1, 150)));
plo = append(plo, param('Nolap', -1));
plo = append(plo, param('Order', 0));

%-------------------------------------------------------------------------
% Compute all TF combinations
function varargout = mlcpsd(varargin)

% MLCPSD
%
% Txy = mlcpsd(X,f,r,m,L,fs,win,order,olap,Lmin)
%
% M Hewitson 19-05-07
%
```

```
%

% Get inputs
X     = varargin{1};
f     = varargin{2};
r     = varargin{3};
m     = varargin{4};
L     = varargin{5};
fs    = varargin{6};
win   = varargin{7};
order = varargin{8};
olap  = varargin{9};
Lmin  = varargin{10};

% -- Set some variables
twopi      = 2.0*pi;
si         = size(X);
nc         = si(1);
nf         = length(f);
Txy        = zeros(nc,nc,nf);
disp_each  = round(nf/100)*10;
minReached = 0;

% Loop over frequencies
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.name
      case 'Kaiser'
        win = specwin(win.name, l, win.psll);
      otherwise
        win = specwin(win.name, 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
    disp(sprintf('++ computing frequency %04d of %d: %f Hz', fi, nf, f(fi)));
  end

  % Loop over input channels
```

```
  for j=1:nc
    % loop over output channels
    for k=1:nc
      if k~=j
        % Core cross-DFT part in C-mex file
        xi    = X(j,:);
        xo    = X(k,:);
        [A, XX, YY, nsegs] = ltpda_dft(xi, xo, l, C, olap, order);
      else
        % Core DFT part in C-mex file
        xi    = X(j,:);
        [A, nsegs] = ltpda_dft(xi, l, C, olap, order);
      end
      S1      = win.ws;
      S2      = win.ws2;
      Txy(j,k,fi)  = 2.0*A/nsegs/fs/S2;
    end % End loop over output channels
  end % End loop over input channels
end % End loop over frequencies

varargout{1} = Txy;

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

retrieve

PURPOSE ^

RETRIEVE param objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE param objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE param objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE param objects with given ids from the LTPDA repository specified by
the input
```

```
% database connection.
%
%%%
```

```
0056
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for param objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for param objects.

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

 DESCRIPTION: EQ overloads the == operator for param objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input param objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two param objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.7 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_legendAdd

PURPOSE ^

LTPDA_LEGENDADD Add a string to the current legend.

SYNOPSIS ^

function ltpda_legendAdd(varargin)

DESCRIPTION ^

```
 LTPDA_LEGENDADD Add a string to the current legend.

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

 DESCRIPTION: LTPDA_LEGENDADD Add a string to the current legend.

 CALL:        legendAdd(fig, 'string')
              plot (curr_axes_handle, history, arg)

 INPUT:       fig    = figure handle (gcf, for example)
              string = string to add

 VERSION:     $Id: ltpda_legendAdd.m,v 1.4 2007/06/07 08:56:24 ingo Exp $

 HISTORY: 26-01-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpda_legendAdd(varargin)
% LTPDA_LEGENDADD Add a string to the current legend.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
% DESCRIPTION: LTPDA_LEGENDADD Add a string to the current legend.
%
% CALL:        legendAdd(fig, 'string')
%              plot (curr_axes_handle, history, arg)
%
% INPUT:       fig    = figure handle (gcf, for example)
%              string = string to add
%
% VERSION:     $Id: ltpda_legendAdd.m,v 1.4 2007/06/07 08:56:24 ingo Exp $
%
% HISTORY: 26-01-07 M Hewitson
%             Creation
%
%%%
```

```
    tag = get(ch, 'Tag');

    if ~strcmp(tag, '')

      str = strvcat(str, tag);
    end
  end
end

str = strvcat(str, strin);

legend(str);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

retrieve

PURPOSE ^

RETRIEVE plist objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE plist objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE plist objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE plist objects with given ids from the LTPDA repository specified by
the input
```

```
% database connection.
%
%%%
```

```
0056
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 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

              The following call returns a parameter list object that
              contains the default parameter values:

              >> pl = ltpda_lincom('Params')

 HISTORY:     14-02-07 M Hewitson
                 Creation.

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = ltpda_lincom(varargin)
% LTPDA_LINCOM make the linear combination of the input analysis objects.
%
%%%
```

```
  if strcmp(in, 'Params')
    varargout{1} = getDefaultPL();
    return
  end
end

% capture input variable names
invars = {};
for j=1:nargin
  invars = [invars cellstr(inputname(j))];
end

ALGONAME = mfilename;
VERSION  = '$Id: ltpda_lincom.m,v 1.13 2007/11/09 14:07:39 ingo Exp $';

% Initialise output
b  = [];
ho = [];
%% Look at inputs

% check for plist and count
% AOs with fsdata and tsdata
pl     = [];
coeffs = [];
na = 0;
for j=1:nargin
  a = varargin{j};
  if isa(a, 'plist')
    pl = a;
  end
end
% if a cdata is supplied, use
% this instead of a plist
as = [];
for j=1:nargin
  a = varargin{j};
  for k=1:length(a)
    ak = a(k);
    if isa(ak, 'ao')
      d = ak.data;
      % is this a constant data
      if isa(d, 'cdata')
        coeffs = d.y;
        % add history of this AO
        ho = [ho ak.hist];
%         % Add coefficients to plist for history purposes
%         pl = plist(param('coeffs', coeffs));
      elseif isa(d, 'tsdata')
        as = [as ak];
      end
    end
  end
```

```
end

% If a plist is supplied and no cdata, use plist
if ~isempty(pl) && isempty(coeffs)
  coeffs = find(pl, 'coeffs');
end

na = length(as);
nc = length(coeffs);

if na < 2
  error('### at least two analysis objects are needed to form a linear combination.');
end

if na ~= nc
  disp(sprintf('Num AOs input: %d', na));
  disp(sprintf('Num Coeffs input: %d', nc));
  error('### specify one coefficient per analysis object.');
end

% loop over all inputs AOs
nameStr = [];
y = 0;
for j=1:na

  % get this data
  aa = as(j);
  da = aa.data;
  dainfo = whos('da');
  % store the first AO data type to
  % check against the others
  if j==1
    d = da;
    dinfo = whos('d');
  end
  % check we know what type of data this is
  if isa(da, 'tsdata') || isa(da, 'fsdata')
    ya = da.y;
  else
    error('### unknown data type.');
  end

  % check against first AO data
  if ~strcmp(dainfo.class, dinfo.class)
    error('### all analysis objects should have the same data type.');
  end

  % All time-series vectors should be the same sample rate
  if isa(d, 'tsdata')
    if da.fs ~= d.fs
      error('### all analysis objects should contain same sample rate data.');
```

```
    end
  end

  % All vectors need to be the same length
  if y~=0
    if length(y) ~= length(ya)
      error('### all vectors should be the same length.');
    end
  end
  % get the appropriate coefficient and add to linear combination
  c = coeffs(j);
  y = y + c.*ya;
  % add name to output str
  if j<=length(invars)
    if isempty(invars{j})
      n = aa.name;
    else
      n = invars{j};
    end
  else
    n = aa.name;
  end
  nameStr = [nameStr '+' num2str(c) sprintf('*%s', n)];

  % add history of this AO
  ho = [ho aa.hist];
end

% clean up nameStr
if nameStr(1) == '+'
  nameStr = nameStr(2:end);
end

% Make output analysis object

% create new output data
if isa(d, 'tsdata')
  data = tsdata(d.x, y);
  data = set(data, 'name', nameStr);
elseif isa(d, 'fsdata')
  data = fsdata(d.x, y, d.fs);
  data = set(data, 'name', nameStr);
else
  error('### unknown data type.');
end

% create new output history
h = history(ALGONAME, VERSION, pl, ho);
h = set(h, 'invars', invars);

% make output analysis object
b = ao(data, h);
```

```

% set name
b = set(b, 'name', nameStr);

varargout{1} = b;

%--------------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

plo = plist();


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

retrieve

PURPOSE ^

RETRIEVE pole objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE pole objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE pole objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE pole objects with given ids from the LTPDA repository specified by
the input
```

```
% database connection.
%
%%%
```

```
0056
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo Exp $
%

```

```
v = 0.54 - 0.46 * cos(2*pi*z);

% Make window struct
w.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for pole objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for pole objects.

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

 DESCRIPTION: EQ overloads the == operator for pole objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input pole objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two pole objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.5 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 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 1.8 2007/11/08 10:24:54 hewitson Exp $

 HISTORY:    14-05-2007 M Hewitson
               Creation

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = ltpda_linedetect('Params')

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ltpda_nfest LTPDA_NFEST Estimate the noise floor of the given frequency-series AO.

This function is called by:

SUBFUNCTIONS ^

- function lines = findLines(ay, f, nay, thresh, N, fsearch)
- function pl = getDefaultPL()

SOURCE CODE ^

```
function varargout = ltpda_linedetect(varargin)
% LTPDA_LINEDETECT find spectral lines in the ao/fsdata objects.
%
%%%
```

```
invars = {};
for j=1:nargin
  if isa(varargin{j}, 'ao')
    invars = [invars cellstr(inputname(j))];
  end
end

ALGONAME = mfilename;
VERSION  = '$Id: ltpda_linedetect.m,v 1.8 2007/11/08 10:24:54 hewitson Exp $';

% Get inputs
as      = [];
ps      = [];
queries = [];
for j=1:nargin
  if isa(varargin{j}, 'ao')
    as = [as varargin{j}];
  end
  if isa(varargin{j}, 'plist')
    ps = [ps varargin{j}];
  end
end

Na = length(as);
if isempty(as)
  error('### Please input at least one AO.');
end

% Combine plists
if ~isempty(ps)
  pl = combine(ps, getDefaultPL());
else
  pl = getDefaultPL();
end

% Get parameters from plist
N       = find(pl, 'N');
fsearch = find(pl, 'fsearch');
thresh  = find(pl, 'thresh');
bw      = find(pl, 'bw');
hc      = find(pl, 'hc');

% Loop over input AOs
bo = [];
for j=1:Na

  a = as(j);
  d = a.data;

  if isa(d, 'fsdata')

    % Make noise-floor estimate
```

```
    nf = ltpda_nfest(a, pl);

    % Make ratio
    r = a./nf;

    % find lines
    lines = findLines(d.y, get(r.data, 'x'), get(r.data, 'y'), thresh, N, fsearch);

    % Make output data
    fs = fsdata([lines(:).f], [lines(:).a], d.fs);

  else
    error('### I can only find lines in frequency-series AOs.');
  end

  %------- Make output AO

  % create new output history
  h = history(ALGONAME, VERSION, pl, a.hist);
  h = set(h, 'invars', invars);

  % make output analysis object
  b = ao(fs, h);

  % set name
  % name for this object
  if isempty(invars{j})
    n1 = a.name;
  else
    n1 = invars{j};
  end
  b = setnh(b, 'name', sprintf('lines(%s)', n1));

  % Add to output array
  bo = [bo b];

end

% return output
varargout{1} = bo;

%-----------------------------------------------------------------------------
% 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



%-----------------------------------------------------------------------
% get default parameter list
function pl = getDefaultPL()

pl = plist();
pl = append(pl, param('N', 10));
pl = append(pl, param('fsearch', [0 1e10]));
pl = append(pl, param('thresh', 2));
pl = append(pl, param('bw', 20));
pl = append(pl, param('hc', 0.8));



% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

retrieve

PURPOSE ^

RETRIEVE provenance objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE provenance objects with given ids from the LTPDA repository specified by
the input
 database connection.

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

 DESCRIPTION: RETRIEVE provenance objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:09 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE provenance objects with given ids from the LTPDA repository
specified by the input
```

```
% database connection.
%
%%%
```

```
0056
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo Exp $
%

```

```
v = 0.5 * (1 - cos(2*pi*z));

% Make window struct
w.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for provenance objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for provenance objects.

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

 DESCRIPTION: EQ overloads the == operator for provenance objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input provenance objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two provenance objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.6 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 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.
%%%
```

```
The following call returns a parameter list object that contains the
default parameter values:

>> pl = ltpda_lpsd('Params')

The following call returns a string that contains the routine CVS version:

>> version = ltpda_lpsd('Version')

The following call returns a string that contains the routine category:

>> category = ltpda_lpsd('Category')

References:
    "Improved spectrum estimation from digitized time series
     on a logarithmic frequency axis", Michael Troebs, Gerhard Heinzel,
     Measurement 39 (2006) 120-129.

M Hewitson 02-02-07

$Id: ltpda_lpsd.m,v 1.19 2008/02/19 19:19:26 hewitson Exp $

%%%
```

```
%   Inputs:
%     as  - array of analysis objects
%     pl  - parameter list (see below)
%
%   Outputs:
%     bs  - array of analysis objects, one for each input
%
%   Parameter list:
%     Kdes  - desired number of averages   (default 100)
%     Lmin  - minimum segment length   (default 0)
%     Jdes  - number of spectral frequencies to compute (default fs/2)
%     win   - a specwin window object
%             Only the design parameters of the window object are used; the
%             window is recomputed for each DFT length inside the lpsd_core
%             algorithm.
%     Olap - desired overlap percentage (default: taken from window)
%     Order - order of detrending
%             -1 - no detrending
%              0 - subtract mean
%              1 - subtract linear fit
%              N - subtract fit of polynomial, order N
%     Scale - Scaling of output. Choose from:
%               AS  - Amplitude (linear) Spectrum
%               ASD - Amplitude (linear) Spectral Density
%               PS  - Power Spectrum
%               PSD - Power Spectral Density [default]
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = ltpda_lpsd('Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = ltpda_lpsd('Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = ltpda_lpsd('Category')
%
% References:
%     "Improved spectrum estimation from digitized time series
%      on a logarithmic frequency axis", Michael Troebs, Gerhard Heinzel,
%      Measurement 39 (2006) 120-129.
%
% M Hewitson 02-02-07
%
% $Id: ltpda_lpsd.m,v 1.19 2008/02/19 19:19:26 hewitson Exp $
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ALGONAME = mfilename;
VERSION  = '$Id: ltpda_lpsd.m,v 1.19 2008/02/19 19:19:26 hewitson 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

% capture input variable names
invars = {};
as     = [];
ps     = [];
for j=1:nargin
  invars = [invars cellstr(inputname(j))];
  if isa(varargin{j}, 'ao')
    as = [as varargin{j}];
  end
  if isa(varargin{j}, 'plist')
    ps = [ps varargin{j}];
  end
end



% check plist
if isempty(ps)
  pl = getDefaultPL();
else
  pl = combine(ps, getDefaultPL);
end

% initialise output array
bs  = [];
bxx = [];

% Go through each input AO
na = length(as);
for i=1:na
  a = as(i);
  d = a.data;

```

```
% check this is a time-series object
if ~isa(d, 'tsdata')
  error('### lpsd requires tsdata (time-series) inputs.');
end

%--- check input parameters
Kdes = find(pl, 'Kdes');   % desired averages
if isempty(Kdes)
  Kdes = 100;
  disp(sprintf('! Using default Kdes of %d', Kdes))
  pl = append(pl, param('Kdes', Kdes));
end

% *** Only use for the old scheduler ***
Kmin = find(pl, 'Kmin');    % minimum averages
if isempty(Kmin)
  Kmin = 1;
  disp(sprintf('! Using default Kmin of %d', Kmin))
  pl = append(pl, param('Kmin', Kmin));
end
% *** Only use for the old scheduler ***

Jdes = find(pl, 'Jdes');    % num desired spectral frequencies
if isempty(Jdes)
  Jdes = round(d.fs/4);
  disp(sprintf('! Using default Jdes of %d', Jdes))
  pl = append(pl, param('Jdes', Jdes));
end
Win = find(pl, 'Win');
if isempty(Win)
  Win = specwin('Kaiser', 100, 250);
  pl = append(pl, param('Win', Win));
end
Nolap = find(pl, 'Nolap');    % desired overlap
if isempty(Nolap) || Nolap == -1
  pl = remove(pl, 'Nolap'); % remove old value
  % use window function rov
  Nolap = Win.rov/100;
  disp(sprintf('! Using recommended Nolap of %d', Nolap))
  pl = append(pl, param('Nolap', Nolap));
end
Order = find(pl, 'Order');    % desired overlap
if isempty(Order)
  Order = 0;
  disp(sprintf('! Using default detrending order of %d', Order))
  pl = append(pl, param('Order', Order));
end
Lmin = find(pl, 'Lmin');
if isempty(Lmin)
  Lmin = 0;
end
Debug = find(pl, 'DEBUG');

```

```
% Get frequency vector
if find(pl, 'OLD SCHEDULER')
  [f,r,m,L,rr,rrr] = ltpda_compute_f(d.fs, length(d.y), Kdes, Kmin, Jdes, Nolap, Debug);
else
  [f, r, m, L, K] = ltpda_ltf_plan(length(d.y), d.fs, Nolap, 1, Lmin, Jdes, Kdes);
end

% compute LPSD
try
  if find(pl, 'M-FILE ONLY')
    % Using pure m-file version
    [P, Pxx, ENBW] = mlpsd(d.y, f, r, m, L, d.fs, Win, Order, Nolap);
  else
    [P, Pxx, ENBW] = mlpsd2(d.y, f, r, m, L, d.fs, Win, Order, Win.rov, Lmin);
  end
catch
  % Using pure m-file version
  [P, Pxx, ENBW] = mlpsd(d.y, f, r, m, L, d.fs, Win, Order, Nolap);
end

% create new output fsdata
scale = find(pl, 'Scale');
switch scale
  case 'AS'
    fs = fsdata(f, sqrt(P), d.fs);
    fs = set(fs, 'yunits', [d.yunits]);
  case 'ASD'
    fs = fsdata(f, sqrt(Pxx), d.fs);
    fs = set(fs, 'yunits', [d.yunits '/\surdHz']);
  case 'PS'
    fs = fsdata(f, P, d.fs);
    fs = set(fs, 'yunits', [d.yunits '^2']);
  case 'PSD'
    fs = fsdata(f, Pxx, d.fs);
    fs = set(fs, 'yunits', [d.yunits '^2/Hz']);
  otherwise
    error(['### Unknown scaling:' scale]);
end
fs = set(fs, 'name', sprintf('lpsd(%s)', d.name));
fs = set(fs, 'enbw', ENBW);

% create new output history
h = history(ALGONAME, VERSION, pl, a.hist);
h = set(h, 'invars', invars);

% make output analysis object
b = ao(fs, h);

% set name
if isempty(invars{i})
```

```
    n = a.name;
  else
    n = invars{i};
  end
  b = setnh(b, 'name', sprintf('lpsd(%s)', n), ...
        'mfilename', ALGONAME);
  bs = [bs b];

end

varargout{1} = bs;

% include AS
% if nargout > 1
%   varargout{2} = bxx;
% end



%-----------------------------------------------------------------------
%
function varargout = mlpsd2(varargin)

% MLPSD2
%
% [S,Sxx,ENBW] = mlpsd2(x,f,r,m,L,fs,win,order,olap)
%
% M Hewitson 19-05-07
%
%


disp('%%%%%%%%   Using ltpda_dft.mex to compute core DFT ');

% Get inputs
x     = varargin{1};
f     = varargin{2};
r     = varargin{3};
m     = varargin{4};
L     = varargin{5};
fs    = varargin{6};
win   = varargin{7};
order = varargin{8};
olap  = varargin{9};
Lmin  = varargin{10};

twopi    = 2.0*pi;

nx   = length(x);
nf   = length(f);
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
    disp(sprintf('++ 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.name
      case 'Kaiser'
        win = specwin(win.name, l, win.psll);
      otherwise
        win = specwin(win.name, 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
    disp(sprintf('   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;


```

```
%-------------------------------------------------------------------------
%
function varargout = mlpsd(varargin)

% MLPSD
%
% [S,Sxx,ENBW] = mlpsd(x,f,r,m,L,fs,win,order,olap)
%
% M Hewitson 19-05-07
%
%

% Get inputs
x     = varargin{1};
f     = varargin{2};
r     = varargin{3};
m     = varargin{4};
L     = varargin{5};
fs    = varargin{6};
win   = varargin{7};
order = varargin{8};
olap  = varargin{9};

twopi    = 2.0*pi;

nx   = length(x);
nf   = length(f);
Sxx  = zeros(nf,1);
S    = zeros(nf,1);
ENBW = zeros(nf,1);

disp_each = round(nf/100)*10;

for j=1:nf

  if mod(j,disp_each) == 0 || j == 1 || j == nf
    disp(sprintf('++ computing frequency %d of %d: %f Hz', j, nf, f(j)));
  end

  % compute DFT exponent and window
  l = L(j);
  switch win.name
    case 'Kaiser'
      win = specwin(win.name, l, win.psll);
    otherwise
      win = specwin(win.name, l);
  end

  p     = 1i * twopi * m(j)/l.*[0:l-1];
  C     = win.win .* exp(p);
  Xolap = (1-olap);
  % do segments
  A  = 0.0;
```

```
  start = 1;
  nsegs = 0;
  while start + l - 1 <= nx

    % get segment
    xs  = x(start:start+l-1);

    % detrend segment
    switch order
      case -1
        % do nothing
      case 0
        xs = xs - mean(xs);
      case 1
        xs = detrend(xs);
      otherwise
        xs = polydetrend(xs, order);
    end

    % make DFT
    a   = C*xs;
    A   = A + a*conj(a);

    % make next step
    start = floor(start + l*Xolap);
    nsegs = nsegs + 1;
  end

  if mod(j,disp_each) == 0 || j == 1 || j == nf
    disp(sprintf('   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;



%-------------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

disp('* creating default plist...');
```

```
plo = plist('Kdes', 100, ...
  'Kmin', 1, ...
  'Lmin', 0, ...
  'Jdes', 100, ...
  'Win', specwin('Kaiser', 10, 200), ...
  'Olap', -1, ...
  'Order', -1, ...
  'Scale', 'PSD');

disp('* done.');

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

retrieve

PURPOSE ^

RETRIEVE pzmodel objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE pzmodel objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE pzmodel objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE pzmodel objects with given ids from the LTPDA repository specified
by the input
```

```
% database connection.
%
%%%
```

```
0056
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo Exp $
%

```

```
z = z * 2 * pi;
v = 0.375 - 0.5 * cos (z) + 0.125 * cos (2 * z);

% Make window struct
w.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for pzmodel objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for pzmodel objects.

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

 DESCRIPTION: EQ overloads the == operator for pzmodel objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input pzmodel objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two pzmodel objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.7 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_ltf_plan

PURPOSE ^

LTPDA_LTF_PLAN computes all input values needed for the LPSD and LTFE

SYNOPSIS ^

function varargout = ltpda_ltf_plan(varargin)

DESCRIPTION ^

```
 LTPDA_LTF_PLAN computes all input values needed for the LPSD and LTFE
 algorithms.
%%%
```

```
The following call returns a string that contains the routine CVS version:

>> version = ltpda_ltf_plan('Version')

The following call returns a string that contains the routine category:

>> category = ltpda_ltf_plan('Category')


REFERENCE:  "lpsd revisited: ltf" / S2-AEI-TN-3052
            2008/02/07  V1.1
            G Heinzel


M Hewitson 19-02-08

$Id: ltpda_ltf_plan.m,v 1.1 2008/02/19 09:11:54 hewitson Exp $

%%%
```

```
%   >> [f, r, b, L, K] = ltpda_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:
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = ltpda_ltf_plan('Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = ltpda_ltf_plan('Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = ltpda_ltf_plan('Category')
%
%
% REFERENCE:  "lpsd revisited: ltf" / S2-AEI-TN-3052
%             2008/02/07  V1.1
%             G Heinzel
%
%
% M Hewitson 19-02-08
%
% $Id: ltpda_ltf_plan.m,v 1.1 2008/02/19 09:11:54 hewitson Exp $
%
%%%
```

```


%% 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

%% ------ 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;
```

```
%%%
```

retrieve

PURPOSE ^

RETRIEVE specwin objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE specwin objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE specwin objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE specwin objects with given ids from the LTPDA repository specified
by the input
```

```
% database connection.
%
%%%
```

```
0056
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo Exp $
%

```

```
z = z * 2 * pi;
v = 0.40897 - 0.5 * cos (z) + 0.09103 * cos (2 * z);

% Make window struct
w.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for specfin objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for specfin objects.

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

 DESCRIPTION: EQ overloads the == operator for specfin objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input specfin objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two specfin objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.5 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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 1.15 2008/02/21 08:52:17 hewitson Exp $

%%%
```

```
%   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 1.15 2008/02/21 08:52:17 hewitson Exp $
%
%%%
```

```
  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, ps, invars] = collect_inputs(varargin, in_names);

% initialise output array
bs = [];

% check plist
if isempty(ps)
  pl = getDefaultPL();
else
  pl = combine(ps, getDefaultPL);
end

% Go through each input AO and build a matrix of signals
na    = length(as);

%---------------- Resample all AOs

disp('*** Resampling AOs...');
fsmax = findFsMax(as);
fspl  = plist(param('fsout', fsmax));
bs = [];
for i=1:na
  a = as(i);
  % check this is a time-series object
  if ~isa(a.data, 'tsdata')
    error('### ltpda_ltfe requires tsdata (time-series) inputs.');
  end
  % Check Fs
  if a.data.fs ~= fsmax
    warning(sprintf('!!! Resampling AO %s/%s to %f Hz', a.name, a.data.name, fsmax));
    a = resample(a, fspl);
  end
  bs = [bs a];
end
as = bs;
na = length(as);
clear bs;
disp('*** Done.');

%---------------- Truncate all vectors

% Get shortest vector
disp('*** Truncating all vectors...');
```

```
lmin = findShortestVector(as);
nsecs = lmin / fsmax;
bs = [];
for i=1:na
  a = as(i);
  if len(a) ~= lmin
    warning(sprintf('!!! Truncating AO %s/%s to %d secs', a.name, a.data.name, nsecs));
    bs = [bs  select(a, 1:lmin)];
  else
    bs = [bs a];
  end
end
as = bs;
clear bs;
disp('*** Done.');


%----------------- Build signal Matrix
N     = len(as(1)); % length of first signal
iS    = zeros(na, N);
for j=1:na
  a = as(j);
  iS(j,:) = a.data.y;
end

%----------------- check input parameters

% Averaging
Kdes = find(pl, 'Kdes');   % desired averages
Kmin = find(pl, 'Kmin');    % minimum averages
Jdes = find(pl, 'Jdes');    % num desired spectral frequencies
if Jdes == -1
  Jdes = round(fsmax/4);
  disp(sprintf('! Using default Jdes of %d', Jdes))
  pl = pset(pl, 'Jdes', Jdes);
end
Win   = find(pl, 'Win');   % Window object
Nolap = find(pl, 'Nolap');    % desired overlap
if Nolap == -1
  % use window function rov
  Nolap = Win.rov/100;
  disp(sprintf('! Using recommended overlap of %d', Nolap))
  pl = pset(pl, 'Nolap', Nolap);
end
Order = find(pl, 'Order');    % desired overlap
if isempty(Order)
  Order = 0;
  disp(sprintf('! Using default detrending order of %d', Order))
  pl = append(pl, param('Order', Order));
end
Lmin = find(pl, 'Lmin');  % Minimum segment length
if isempty(Lmin)
```

```
  Lmin = 0;
end

%----------------- Get frequency vector
if find(pl, 'OLD SCHEDULER')
  [f,r,m,L,rr,rrr] = ltpda_compute_f(fsmax, lmin, Kdes, Kmin, Jdes, Nolap, 0);
else
  [f, r, m, L, K] = ltpda_ltf_plan(lmin, fsmax, Nolap, 1, Lmin, Jdes, Kdes);
end

%----------------- compute TF Estimates

Txy = mltfe(iS, f, r, m, L, fsmax, Win, Order, Nolap*100, Lmin);

%----------------- Build output Matrix of AOs
disp('** Building output AO matrix')
bs  = cell(na);
for j=1:na  % Loop over input signal
  bi = as(j);
  for k=1:na  % Loop over output signal
    bo = as(k);
    % create new output fsdata
    fs = fsdata(f, squeeze(Txy(j,k,:)), fsmax);
    fs = set(fs, 'name', sprintf('LTFE(%s->%s)', bi.data.name, bo.data.name));
    fs = set(fs, 'yunits', [bo.data.yunits '/' bi.data.yunits]);
    % create new output history
    h = history(ALGONAME, VERSION, pl, [bi.hist bo.hist]);
    try
      h = set(h, 'invars', invars([j k]));
    catch
      warning('!!! Unable to set input variable names for history. Perhaps you input a vector of AOs.');
    end
    % make output analysis object
    b = ao(fs, h);
    % set name
    try
      if isempty(invars{k})
        nk = bo.name;
      else
        nk = invars{k};
      end
      if isempty(invars{j})
        nj = bi.name;
      else
        nj = invars{j};
      end
    catch
      warning('!!! Unable to set input variable names for history. Perhaps you input a vector of AOs.');
      nk = bo.name;
      nj = bi.name;
```

```
    end

    b = setnh(b, 'name', sprintf('LTFE(%s->%s)', nj, nk));
    % add to outputs
    bs(j,k) = {b};

  end % End loop over output signal
end % End loop over input signal

% now we have a cell matrix of AOs but
% we want a normal matrix
b  = [bs{:}];
bs = reshape(b, na, na);
varargout{1} = bs;

%-----------------------------------------------------------------------
% 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


%-----------------------------------------------------------------------
% 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

%-----------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('Kdes', 100));
plo = append(plo, param('Jdes', 1000));
plo = append(plo, param('Kmin', 2));
plo = append(plo, param('Lmin', 0));
plo = append(plo, param('Win',  specwin('Kaiser', 1, 150)));
plo = append(plo, param('Nolap', -1));
plo = append(plo, param('Order', 0));

```

```

%-----------------------------------------------------------------------
% Compute all TF combinations
function varargout = mltfe(varargin)

% MLTFE
%
% Txy = mltfe(X,f,r,m,L,fs,win)
%
% M Hewitson 19-05-07
%
%

% Get inputs
X     = varargin{1};
f     = varargin{2};
r     = varargin{3};
m     = varargin{4};
L     = varargin{5};
fs    = varargin{6};
win   = varargin{7};
order = varargin{8};
olap  = varargin{9};
Lmin  = varargin{10};


% --- Prepare some variables
twopi      = 2.0*pi;
si         = size(X);
nc         = si(1);
nf         = length(f);
Txy        = zeros(nc,nc,nf);
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.name
      case 'Kaiser'
        win = specwin(win.name, l, win.psll);
      otherwise
        win = specwin(win.name, 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
    disp(sprintf('++ computing frequency %d of %d: %f Hz', fi, nf, f(fi)));
  end

  % Loop over input channels
  for j=1:nc
    % loop over output channels
    for k=1:nc
      if k~=j
        % Core cross-DFT part in C-mex file
        xi    = X(j,:);
        xo    = X(k,:);
        % We need cross-spectrum and Power spectrum
        [XY, XX, YY, nsegs] = ltpda_dft(xi, xo, l, C, olap, order);
        Txy(j,k,fi)  = XY/XX;
      else
        % Trivial case of equal input and output
        Txy(j,k,fi)  = 1;
      end
    end % End loop over output channels
  end % End loop over input channels
end % End loop over frequencies

% Set output
varargout{1} = Txy;

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

retrieve

PURPOSE ^

RETRIEVE time objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE time objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE time objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:07 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE time objects with given ids from the LTPDA repository specified by
the input
```

```
% database connection.
%
%%%
```

```
0056
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo Exp $
%

```

```
z = z * 2 * pi;
v = 0.4243801 - 0.4973406 * cos (z) + 0.0782793 * cos (2 * z);

% Make window struct
w.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for time objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for time objects.

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

 DESCRIPTION: EQ overloads the == operator for time objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input time objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two time objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.7 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_lxspec

PURPOSE ^

LTPDA_LXSPEC performs cross-spectral analysis of various forms.

SYNOPSIS ^

function varargout = ltpda_lxspec(varargin)

DESCRIPTION ^

```
 LTPDA_LXSPEC performs cross-spectral analysis of various forms.

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

 DESCRIPTION: LTPDA_LXSPEC 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_ltfe).

 CALL:              Y = ltpda_lxspec(X, f, r, m, L, fs, win, order, method);
               [Y,Ys] = ltpda_lxspec(X, f, r, m, L, fs, win, order, method);

 INPUTS:     X       - a matrix of input data samples
             f,r,m,L - outputs of ltpda_compute_f
             fs      - sample rate of data vectors
             win     - specwin object used to window the data
             order   - order of detrending:
                       -1 - no detrending
                        0 - subtract mean
                        1 - subtract linear fit
                        N - subtract fit of polynomial, order N
             method  - A collection of 'Xpsd', 'TFE', 'Cohere'

 OUTPUTS:    Y    - A cell array containing one NxN matrix of vectors for each
                    of the requested estimate types
             Ys   - A cell array of NxN matrices of variance estiamtes for Y

 VERSION:    $Id: ltpda_xspec.m,v 1.7 2007/10/08 11:28:29 hewitson Exp $

 HISTORY:    28-10-2007 M Hewitson
                Creation


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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function varargout = ltpda_lxspec(varargin)
% LTPDA_LXSPEC performs cross-spectral analysis of various forms.
%
%%%
```

```
f     = varargin{2};
r     = varargin{3};
m     = varargin{4};
L     = varargin{5};
fs    = varargin{6};
win   = varargin{7};
order = varargin{8};

% do we need to compute variance?
if nargout == 2
  computeVariance = 1;
else
  computeVariance = 0;
end

twopi    = 2.0*pi;

si   = size(X);
nx   = si(1);
nc   = si(2);
nf   = length(f);
Txy  = zeros(nc,nc,nf);
Sig  = zeros(nc,nc,nf);
ENBW = zeros(nc,nc,nf);

switch order
  case -1
    detrender = '';
  case 0
    detrender = 'xi = xi - mean(xi);xo = xo - mean(xo);';
  case 1
    detrender = 'xi = detrend(xi);xo = detrend(xo);';
  otherwise
    detrender = 'xi = polydetrend(xi, order); xo = polydetrend(xo, order);';
end

disp_each = round(nf/100)*10;
for fi=1:nf

  % compute DFT exponent and window
  l = L(fi);
  switch win.name
    case 'Kaiser'
      win = specwin(win.name, l, win.psll);
    otherwise
      win = specwin(win.name, l);
  end

  p     = 1i * twopi * m(fi)/l.*[0:l-1];
  C     = win.win .* exp(p);
  Xolap = (1-win.rov/100);
  if mod(fi,disp_each) == 0 || fi == 1 || fi == nf
```

```
    disp(sprintf('++ computing frequency %03d of %03d: %f Hz', fi, nf, f(fi)));
  end

  %-------------- Count segments
  nsegs = 0;
  start = 1;
  lm1   = l-1;
  while start + l - 1 <= nx
    start = floor(start + l*Xolap);
    nsegs = nsegs + 1;
  end
  % now allocate indices for segments
%   idxs  = repmat(struct('idx', zeros(1,l)), 1, nsegs);
  idxs  = zeros(nsegs,l);
  nsegs = 0;
  start = 1;
  while start + l - 1 <= nx
    idxs(1+nsegs,:) = start:start+lm1;
    start = floor(start + l*Xolap);
    nsegs = nsegs + 1;
  end
  Sxy = zeros(nsegs,1);
  i2  = zeros(nsegs,1);
  xi  = zeros(l,1);
  xo  = zeros(l,1);

  %----------------- Process the channels

  % Loop over input channels
  for j=1:nc
    % loop over output channels
    for k=1:nc
      % compute lower triangle of matrix
      if j<k

        % do segments
        Sxx    = 0;
        Syy    = 0;
        Sxy(:) = 0;
        i2(:)  = 0;

        % Loop over segments
        for s=1:nsegs
          % get segments
          xi(:) = X(idxs(s,:),j);
          xo(:) = X(idxs(s,:),k);

          % detrend segments
          eval(detrender);

          % compute DFT
          ti      = C*xi;
```

```
          to      = C*xo;
          cto     = conj(to);
          Sxy(s)  = cto*ti;
          Sxx     = Sxx + ti*conj(ti);
          Syy     = Syy + to*cto;

        end % End segment loop
        disp(['-processed ' num2str(nsegs) ' segments']);

        S1      = sum(win.ws);
        S12     = S1*S1;
        S2      = win.win*win.win';
        ENBW(j,k,fi) = fs*S2/S12;
        Txy(j,k,fi)  = (abs(sum(Sxy))^2)/(Sxx*Syy);

        %----------- compute variance
        if computeVariance
          A     = 0;
          scale = fs*S2;
          for s=1:nsegs
            S  = (Sxy(s)/scale-Txy(j,k,fi))^2;
            A  = A + S * i2(s);
          end
          Sig(j,k,fi) = A/sum(i2);
        end % End compute variance

      end % End if j<k
    end % End loop over output channels
  end % End loop over input channels
end % End loop over frequencies

% Fill the holes
aoOnes = ones(1,nf);
for j=1:nc
  for k=1:nc
    if j == k % the answer is 1
      Txy(j,k,:) = aoOnes;
    end
    if j>k
      Txy(j,k,:) = Txy(k,j,:);
      if computeVariance
        Sig(j,k,:) = Sig(k,j,:);
      end
    end
  end
end


varargout{1} = Txy;
if nargout == 2
  varargout{2} = ENBW;
elseif nargout == 3
```

```
  varargout{3} = Sig;
  varargout{2} = ENBW;
end
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

retrieve

PURPOSE ^

RETRIEVE timeformat objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE timeformat objects with given ids from the LTPDA repository specified by
the input
 database connection.

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

 DESCRIPTION: RETRIEVE timeformat objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE timeformat objects with given ids from the LTPDA repository
specified by the input
```

```
% database connection.
%
%%%
```

```
0056
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:03 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:03 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for timeformat objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for timeformat objects.

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

 DESCRIPTION: EQ overloads the == operator for timeformat objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input timeformat objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two timeformat objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.5 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_mat2str

PURPOSE ^

LTPDA_MAT2STR overloads the mat2str operator to set the precision at a central place.

SYNOPSIS ^

function str = ltpda_mat2str(number)

DESCRIPTION ^

```
 LTPDA_MAT2STR overloads the mat2str operator to set the precision at a central
place.

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

 DESCRIPTION: LTPDA_MAT2STR overloads the mat2str operator to set the
              precision at a central place.

 CALL:        str = ltpda_mat2str(number);
              str = ltpda_mat2str(matrix);

 VERSION:     $Id: ltpda_mat2str.m,v 1.5 2008/02/19 17:28:38 ingo Exp $

 HISTORY:     26-07-2007 Diepholz
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function str = ltpda_mat2str(number)
% LTPDA_MAT2STR overloads the mat2str operator to set the precision at a
central place.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
% DESCRIPTION: LTPDA_MAT2STR overloads the mat2str operator to set the
%               precision at a central place.
%
% CALL:         str = ltpda_mat2str(number);
%               str = ltpda_mat2str(matrix);
%
% VERSION:      $Id: ltpda_mat2str.m,v 1.5 2008/02/19 17:28:38 ingo Exp $
%
% HISTORY:      26-07-2007 Diepholz
%                  Creation
%
%%%
```

retrieve

PURPOSE ^

RETRIEVE timespan objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE timespan objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE timespan objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE timespan objects with given ids from the LTPDA repository specified
by the input
```

```
% database connection.
%
%%%
```

```
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:02 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:02 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for timespan objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for timespan objects.

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

 DESCRIPTION: EQ overloads the == operator for timespan objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input timespan objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two timespan objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.5 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_matchvectors

PURPOSE ^

LTPDA_RESAMPLE make the two vectors have the same length.

SYNOPSIS ^

function [x, y, fs] = ltpda_matchvectors(x, y, xfs, yfs)

DESCRIPTION ^

```
 LTPDA_RESAMPLE make the two vectors have the same length.

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

 DESCRIPTION: LTPDA_RESAMPLE make the two vectors have the same length and
              sample rate by down-sampling the one with the higher samplerate.
              Vectors are also truncated to the length of the shorter vector.

 CALL:       [xo, yo, fs] = mresample(x, y, xfs, yfs)

 INPUTS:     x   - data vector
             y   - data vector
             xfs - sample rate of x
             yfs - sample rate of y

 OUTPUTS:    x  - new data vector
             y  - new data vector
             fs - new sample rate of x AND y

 VERSION: $Id: ltpda_matchvectors.m,v 1.3 2007/08/14 09:25:24 ingo Exp $

 HISTORY: 26-01-2007 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function [x, y, fs] = ltpda_matchvectors(x, y, xfs, yfs)
% LTPDA_RESAMPLE make the two vectors have the same length.
%
%%%
```

```
end
if ny > nx
  dv_disp('!!! vectors are not equal lengths. Truncating y. !!!');
  y = y(1:nx);
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

retrieve

PURPOSE ^

RETRIEVE tsdata objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE tsdata objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE tsdata objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:09 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE tsdata objects with given ids from the LTPDA repository specified by
the input
```

```
% database connection.
%
%%%
```

```
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:03 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:03 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for tsdata objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for tsdata objects.

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

 DESCRIPTION: EQ overloads the == operator for tsdata objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input tsdata objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two tsdata objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.9 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_msuptitle

PURPOSE

LTPDA_MSUPTITLE Puts a title above all subplots.

SYNOPSIS

function hout = ltpda_msuptitle(str)

DESCRIPTION

```
 LTPDA_MSUPTITLE Puts a title above all subplots.

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

 DESCRIPTION: LTPDA_MSUPTITLE Puts a title above all subplots.
                 MSUPTITLE('text') adds text to the top of the figure
                 above all subplots (a "super title"). Use this function
              after all subplot commands.

 CALL:        ltpda_xaxis(x1,x2)

 REMARK:      If the figure or axis units are non-default, this will break.

 VERSION:     $Id: ltpda_msuptitle.m,v 1.4 2007/07/16 12:52:20 ingo Exp $

 HISTORY:     15-06-1995 Drea Thomas (drea@mathworks.com)
                 Creation.
              13-12-2000 John Cristion
                 Modified.
              13-03-2004 Mark Histed (histed@mit.edu)
                 Fix disappearing legend on last plot
              xx-xx-xxxx M Hewitson
                 Just a copy and rename with some small adjustements

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

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
function hout = ltpda_msuptitle(str)
% LTPDA_MSUPTITLE Puts a title above all subplots.
%
%%%
```

```
        pos = get(gcf,'position');
        set(gcf,'units',figunits);
    else
        pos = get(gcf,'position');
    end
    ff = (fs-4)*1.27*5/pos(4)*fudge;

        % The 5 here reflects about 3 characters of height below
        % an axis and 2 above. 1.27 is pixels per point.

% Determine the bounding rectange for all the plots

% h = findobj('Type','axes');

% findobj is a 4.2 thing.. if you don't have 4.2 comment out
% the next line and uncomment the following block.

% h = findobj(gcf,'Type','axes');  % Change suggested by Stacy J. Hills

% If you don't have 4.2, use this code instead
ch = get(gcf,'children');
h=[];
for i=1:length(ch),
 if strcmp(get(ch(i),'type'),'axes'),
   h=[h,ch(i)];
 end
end


max_y=0;
min_y=1;

oldtitle =0;
for i=1:length(h),
    if (~strcmp(get(h(i),'Tag'),'suptitle')),
        pos=get(h(i),'pos');
        if (pos(2) < min_y), min_y=pos(2)-ff/5*3;end;
        if (pos(4)+pos(2) > max_y), max_y=pos(4)+pos(2)+ff/5*2;end;
    else
        oldtitle = h(i);
    end
end

if max_y > plotregion,
  scale = (plotregion-min_y)/(max_y-min_y);
  for i=1:length(h),
    get(h(i),'Tag')
    if ~strcmp(get(h(i),'Tag'),'legend')
      pos = get(h(i),'position');
      pos(2) = (pos(2)-min_y)*scale+min_y;
      pos(4) = pos(4)*scale-(1-scale)*ff/5*3;
      set(h(i),'position',pos);
```

```
    end
  end
end

np = get(gcf,'nextplot');
set(gcf,'nextplot','add');
if (oldtitle),
    delete(oldtitle);
end
ha=axes('pos',[0 1 1 1],'visible','off','Tag','suptitle');
ht=text(.5,titleypos-1,str);set(ht,'horizontalalignment','center','fontsize',fs);
set(gcf,'nextplot',np);
axes(haold);

% fix legend if one exists
legH = legend;
if ~isempty(legH)
    axes(legH);
end

if nargout,
    hout=ht;
end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

retrieve

PURPOSE ^

RETRIEVE xydata objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE xydata objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE xydata objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE xydata objects with given ids from the LTPDA repository specified by
the input
```

```
% database connection.
%
%%%
```

```
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:03 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:03 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for xydata objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for xydata objects.

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

 DESCRIPTION: EQ overloads the == operator for xydata objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input xydata objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two xydata objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.8 2008/02/18 14:41:13 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 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]

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = ltpda_nfest('Params')



 VERSION: $Id: ltpda_nfest.m,v 1.7 2007/11/08 10:24:54 hewitson Exp $

 HISTORY: 14-05-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpda_linedetect LTPDA_LINEDETECT find spectral lines in the ao/fsdata objects.

SUBFUNCTIONS ^

- function pl = getDefaultPL()
- function nf = nfest(pxx, bw, hc)

SOURCE CODE ^

```
function varargout = ltpda_nfest(varargin)
% LTPDA_NFEST Estimate the noise floor of the given frequency-series AO.
%
%%%
```

```
    invars = [invars cellstr(inputname(j))];
  end
end

ALGONAME = mfilename;
VERSION  = '$Id: ltpda_nfest.m,v 1.7 2007/11/08 10:24:54 hewitson Exp $';

% Get inputs
as      = [];
ps      = [];
queries = [];
for j=1:nargin
  if isa(varargin{j}, 'ao')
    as = [as varargin{j}];
  end
  if isa(varargin{j}, 'plist')
    ps = [ps varargin{j}];
  end
end

Na = length(as);
if isempty(as)
  error('### Please input at least one AO.');
end

% Combine plists
if ~isempty(ps)
  pl = combine(ps, getDefaultPL());
else
  pl = getDefaultPL();
end

% Get parameters from plist
bw      = find(pl, 'bw');
hc      = find(pl, 'hc');

% Loop over input AOs
bo = [];
for j=1:Na

  a = as(j);
  d = a.data;

  if isa(d, 'fsdata')

    nf = nfest(d.y, bw, hc);

  else
    error('### I can only find lines in frequency-series AOs.');
  end

  %------- Make output AO
```

```

  % make fsdata
  d = set(d, 'y', nf);
  d = set(d, 'name', sprintf('nfest(%s)', d.name));

  % create new output history
  h = history(ALGONAME, VERSION, pl, a.hist);
  h = set(h, 'invars', invars);

  % make output analysis object
  b = ao(d, h);

  % set name
  % name for this object
  if isempty(invars{j})
    n1 = a.name;
  else
    n1 = invars{j};
  end
  b = setnh(b, 'name', sprintf('nfest(%s)', n1));

  % Add to output array
  bo = [bo b];

end

varargout{1} = bo;

%-----------------------------------------------------------------------
% get default parameter list
function pl = getDefaultPL()

pl = plist();
pl = append(pl, param('bw', 20));
pl = append(pl, param('hc', 0.8));


%-----------------------------------------------------------------------
% compute noise floor estimate
function nf = nfest(pxx, bw, hc)

N = length(pxx);
nf = [];

for j=1:N

  % Determine the interval we are looking in
  interval = j-bw/2:j+bw/2;
  idx = find(interval<=0);
  interval(idx)=1;
  idx = find(interval>N);
  interval(idx)=N;
```

```

  % calculate median value of interval
  % after throwing away outliers
  trial = sort(pxx(interval));
  b = round(hc*length(trial));
  nf(j) = median(trial(1:b));

end

nf = nf(:);

% END

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

retrieve

PURPOSE ^

RETRIEVE zero objects with given ids from the LTPDA repository specified by the input

SYNOPSIS ^

function objs = retrieve(conn, ids)

DESCRIPTION ^

```
 RETRIEVE zero objects with given ids from the LTPDA repository specified by the
input
 database connection.

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

 DESCRIPTION: RETRIEVE zero objects with given ids from the LTPDA
              repository specified by the input database connection.

 CALL:        p = retrieve(conn, ids)



 VERSION:     $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function objs = retrieve(conn, ids)

% RETRIEVE zero objects with given ids from the LTPDA repository specified by
the input
```

```
% database connection.
%
%%%
```

```
0056
0057
0058
0059
0060
0061 % END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:03 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:03 ingo Exp $
%

```

```
v = ones(size(z));

% Make window struct
w.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for xyzdata objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for xyzdata objects.

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

 DESCRIPTION: EQ overloads the == operator for xyzdata objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input xyzdata objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two xyzdata objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.3 2008/02/18 14:41:14 ingo Exp $

 HISTORY:     24-12-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_noisegen

PURPOSE ^

LTPDA_NOISEGEN uses pzm2ab and franklin to generate a time-series

SYNOPSIS ^

function varargout = ltpda_noisegen(varargin)

DESCRIPTION ^

```
LTPDA_NOISEGEN uses pzm2ab and franklin to generate a time-series
from a given pzmodel

calls:
>> [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:
       - nsecs
       - fs
       - pzmodel with gain

b   - analysis object containing the resulting timeseries
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
           - gain
           - fs    - sampling frequency
           - nsecs - number of seconds

The following call returns a parameter list object that contains the
default parameter values:
>> pl = ltpda_noisegen('Params')

A Monsky 24-07-07

$Id: ltpda_noisegen.m,v 1.10 2007/11/09 14:07:39 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

- ngconv NGINIT 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

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = ltpda_noisegen(varargin)

% LTPDA_NOISEGEN uses pzm2ab and franklin to generate a time-series
% from a given pzmodel
%
% calls:
% >> [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:
%        - nsecs
%        - fs
%        - pzmodel with gain
%
% b   - analysis object containing the resulting timeseries
% 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
%            - gain
%            - fs    - sampling frequency
%            - nsecs - number of seconds
%
% The following call returns a parameter list object that contains the
% default parameter values:
% >> pl = ltpda_noisegen('Params')
%
```

```
% A Monsky 24-07-07
%
% $Id: ltpda_noisegen.m,v 1.10 2007/11/09 14:07:39 ingo Exp $
%

ALGONAME = mfilename;
VERSION  = '$Id: ltpda_noisegen.m,v 1.10 2007/11/09 14:07:39 ingo Exp $';

% Check if this is a call for parameters
if nargin == 1
  in = char(varargin{1});
  if strcmp(in, 'Params')
    varargout{1} = getDefaultPL();
    return
  end
end

% capture input variable names
invars = {};
ps   = [];
for j=1:nargin
  invars = [invars cellstr(inputname(j))];
  if isa(varargin{j}, 'plist')
    ps = [ps varargin{j}];
  end
end

pl = combine(ps);

% look for all parameters
pzmodel = find(pl, 'pzmodel');
gain    = get(pzmodel, 'gain');
fs      = find(pl, 'fs');
nsecs   = find(pl, 'nsecs');
state   = find(pl, 'state');
Tinit   = find(pl, 'Tinit');
Tprop   = find(pl, 'Tprop');
E       = find(pl, 'E');
num     = find(pl, 'num');
den     = find(pl, 'den');

% get data out of aos
if ~isempty(state)
    d     = state.data;
    state = d.y;
end
if ~isempty(Tinit)
    d     = Tinit.data;
    Tinit = d.y;
end
if ~isempty(Tprop)
    d     = Tprop.data;
    Tprop = d.y;
```

```
end
if ~isempty(E)
    d     = E.data;
    E = d.y;
end
if ~isempty(num)
    d     = num.data;
    num = d.y;
end
if ~isempty(den)
    d     = den.data;
    den = d.y;
end
lengden = length(den);
% check calling state
if isempty(pzmodel)
   if isempty(Tinit)
        error('### There is no Tinit matrix defined plist.');
    end
   gain = find(pl, 'gain');
  % set state vector
  if isempty(state)
    disp('There is no pzmodel and no initial state vector given, so the initial state vector will be calculated by using the given matrices.\\');
    % make initial state vector
    y = nginit(Tinit);
  else
    disp('State vector from input parameter list is used as input state.');
    % use input state
    y = state;
  end

  % propogate to make noise vector
  [x, yo] = ngprop(Tprop, E, num, y, fs*nsecs);
  state = yo;
else
  % conversion
  disp('Filter coefficients are calculated from input pzmodel.');
  [num, den] = ngconv(pzmodel);
  % create matrices
  disp('Matrices are calculated from evaluated denominator coefficients.');
  [Tinit, Tprop, E] = ngsetup(den, fs);

  % set state vector
  if isempty(state)
    disp('Since there is no given state vector it will be calculated.');
    % make initial state vector
    y = nginit(Tinit);
  else
    disp('Since there is a state vector given it is used for further calculations.');
    % use input state
```

```
    y = state;
  end
  % propogate to make noise vector
  [x, yo] = ngprop(Tprop, E, num, y, fs*nsecs);
  state = yo;
end

%----------------- Build outputs
% make a new history object
h = history(ALGONAME, VERSION, pl);
h = set(h, 'invars', invars);
% build variables into data object
t = x*gain;
b = ao(tsdata(t,fs),h);
b = setnh(b, 'name', sprintf('noisegen(%s)', b.name));

state  = ao(cdata(state),h);
Tinit  = ao(cdata(Tinit),h);
Tprop  = ao(cdata(Tprop),h);
E      = ao(cdata(E),h);
num    = ao(cdata(num),h);
den    = ao(cdata(den),h);
if nargout == 2
  % build pl with 'state'
  pl1 = plist('state', state);
end

if nargout == 3
  % build another pl with 'Tinit', 'Tprop', 'E', 'num', 'den'
  pl1 = plist('state',state);
  pl2 = plist('Tinit',Tinit,'Tprop',Tprop,'E',E,'num',num,'den',den,'gain',
gain,'fs',fs,'nsecs',nsecs);
end

%----------------- Pass outputs

if nargout == 1
  varargout{1} = b;
end

if nargout == 2
  varargout{1} = b;
  varargout{2} = pl1;
end

if nargout == 3
  varargout{1} = b;
  varargout{2} = pl1;
  varargout{3} = pl2;
end


% END
```

```
%-------------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

disp('* creating default plist...');
plo = plist();
plo = append(plo,param('nsecs',1000));
disp('* done.');


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

rp2iir

PURPOSE ^

RP2IIR Return a,b coefficients for a real pole designed using the bilinear transform.

SYNOPSIS ^

function [a,b] = rp2iir(p, fs)

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 1.5 2008/02/12 21:38:23 hewitson Exp $

 HISTORY:     03-03-07 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- get GET get pole properties.

This function is called by:

SOURCE CODE ^

```
function [a,b] = rp2iir(p, fs)
% RP2IIR Return a,b coefficients for a real pole designed using the bilinear transform.
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:03 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:03 ingo Exp $
%

```

```
z = z * 2 * pi;
v = 0.26526 - 0.5 * cos (z) + 0.23474 * cos (2 * z);

% Make window struct
w.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

eq

PURPOSE ^

EQ overloads the == operator for zero objects.

SYNOPSIS ^

function result = eq(c1,c2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for zero objects.

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

 DESCRIPTION: EQ overloads the == operator for zero objects.

              All fields are checked.

 CALL:        result = eq(c1,c2)
              result = eq(c1,c2,  exc_list)
              result = eq(c1,c2, 'property1', 'property2')
              result = eq(c1,c2, 'class/property', 'class/property')
              result = eq(c1,c2,  plist('Exceptions', 'exception-list'))

 PLIST:       key:   'Exceptions'
              value:  String or cell-array with exceptions.

 EXAMPLES:    result = eq(c1,c2, 'name', 'created')
              result = eq(c1,c2, 'ao/name')
              result = eq(c1,c2,  plist('Exceptions', 'name'))
              result = eq(c1,c2,  plist('Exceptions', {'name', 'created'}))

 INPUTS:      c1,c2    - input zero objects
              exc_list - exception list
                         List of properties which are not checked.

 OUTPUTS:     If the two zero objects are considered equal, result == 1,
              otherwise, result == 0.

 VERSION:     $Id: eq.m,v 1.5 2008/02/18 14:41:14 ingo Exp $

 HISTORY:     29-08-2007 M Hewitson
                 Creation
```

```
%%%
```

```
CATEGORY      = 'Relational Operator';
DEFAULT_PLIST = plist();

result = generic_eq(c1, c2, varargin, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_num2str

PURPOSE ^

LTPDA_NUM2STR uses sprintf to convert a data vector to a string with a fixed precision.

SYNOPSIS ^

function str = ltpda_num2str(number)

DESCRIPTION ^

```
 LTPDA_NUM2STR uses sprintf to convert a data vector to a string with a fixed
precision.

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

 DESCRIPTION: uses sprintf to convert a data vector to a string with a fixed
precision.

 CALL:        str = ltpda_num2str(number);

 VERSION:     $Id: ltpda_num2str.m,v 1.2 2007/10/26 14:39:54 hewitson Exp $

 HISTORY:     18-10-2007 Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpda_xmlwrite LTPDA_XMLWRITE Add an object to a xml DOM project.

SOURCE CODE ^

```
function str = ltpda_num2str(number)
% LTPDA_NUM2STR uses sprintf to convert a data vector to a string with a fixed
precision.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%
% DESCRIPTION: uses sprintf to convert a data vector to a string with a fixed precision.
%
% CALL:        str = ltpda_num2str(number);
%
% VERSION:     $Id: ltpda_num2str.m,v 1.2 2007/10/26 14:39:54 hewitson Exp $
%
% HISTORY:     18-10-2007 Hewitson
%                 Creation
%
%%%
```

rstruct

PURPOSE ^

RSTRUCT recursively converts an object into a structure. This is the same

SYNOPSIS ^

function s = rstruct(obj)

DESCRIPTION ^

```
RSTRUCT recursively converts an object into a structure. This is the same
behaviour as MATLAB's struct(obj) except that it recursively converts all
sub-objects into structures as well.

M Hewitson 02-06-07

$Id: rstruct.m,v 1.1 2007/09/18 17:44:36 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function s = rstruct(obj)
% RSTRUCT recursively converts an object into a structure. This is the same
% behaviour as MATLAB's struct(obj) except that it recursively converts all
% sub-objects into structures as well.
%
% M Hewitson 02-06-07
%
% $Id: rstruct.m,v 1.1 2007/09/18 17:44:36 hewitson Exp $
%

s = struct(obj);
names = fieldnames(s);
fields = struct2cell(s);

```

```
for j=1:length(names)

  f = fields{j};
  if isobject(f)
    try
      cmd = sprintf('s.%s = rstruct(f);', names{j});
      eval(cmd);
    end
  end
end
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:03 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:03 ingo Exp $
%

```

```
z = z * 2 * pi;
v = 0.28235 - 0.52105 * cos (z) + 0.19659 * cos (2 * z);

% Make window struct
w.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_obj_retrieve

PURPOSE ^

LTPDA_OBJ_RETRIEVE retrieves a collection of objects from an LTPDA repository.

SYNOPSIS ^

function varargout = ltpda_obj_retrieve(varargin)

DESCRIPTION ^

```
 LTPDA_OBJ_RETRIEVE retrieves a collection of objects from an LTPDA repository.


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

 DESCRIPTION: RETRIEVE a collection of objects from an LTPDA repository.

 CALL:    objs    = ltpda_obj_retrieve(conn, obj_id_1, obj_id_2)
          [o1,o2] = ltpda_obj_retrieve(conn, obj_id_1, obj_id_2)
          objs    = ltpda_obj_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.

 VERSION:     $Id: ltpda_obj_retrieve.m,v 1.14 2008/02/24 10:16:58 hewitson Exp $

 HISTORY: 30-08-07 M Hewitson
```

```
              Creation

%%%
```

```
% not packed in a cell array.
%
% VERSION:     $Id: ltpda_obj_retrieve.m,v 1.14 2008/02/24 10:16:58 hewitson Exp $
%
% HISTORY: 30-08-07 M Hewitson
%             Creation
%
%%%
```

```
disp(['*** retrieving objects [' num2str(ids) ']'])

for j=1:length(ids)
  % Get object
  xdoc = mysql_getXdoc(conn, ids(j));
  if ~isempty(xdoc)
    obj = ltpda_xmlread(xdoc);
    if j==1
      objs = {obj};
    else
      objs = [objs {obj}];
    end

    % make transaction entry
    t     = time();
    tdate = t.time_str;
    try
      message = ltpda_insert(conn, 'transactions',...
        'obj_id', ids(j),...
        'user_id', userid,...
        'transdate', tdate,...
        'direction', 'out'...
        );
      disp(sprintf('  + 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
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

rz2iir

PURPOSE ^

RZ2IIR Return a,b IIR filter coefficients for a real zero designed using the bilinear transform.

SYNOPSIS ^

function [a,b] = rz2iir(z, fs)

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 1.6 2008/02/12 19:54:16 hewitson Exp $

 HISTORY:     18-02-2003 Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- get GET get zero properties.

This function is called by:

SOURCE CODE ^

```
function [a,b] = rz2iir(z, fs)
% RZ2IIR Return a,b IIR filter coefficients for a real zero designed using the bilinear transform.
%
%%%
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:03 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:03 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

explore_ao

PURPOSE ^

EXPLORE_AO for exploring analysis objects and plotting/displaying their fields

SYNOPSIS ^

function explore_ao(varargin)

DESCRIPTION ^

```
 EXPLORE_AO for exploring analysis objects and plotting/displaying their fields

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

 DESCRIPTION: EXPLORE_AO 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:    explore_ao;              % Read the ao's from the'base' workspace
          explore_ao(ao);
          explore_ao(ao_vector);
          explore_ao(ao_matrix);

 HINT:    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

 VERSION: $Id: explore_ao.m,v 1.14 2008/02/14 10:41:02 ingo Exp $

 HISTORY: 10-06-07 Diepholz
             Creation

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

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 explore_ao(varargin)
% EXPLORE_AO for exploring analysis objects and plotting/displaying their fields
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: EXPLORE_AO 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:    explore_ao;              % Read the ao's from the'base' workspace
%          explore_ao(ao);
%          explore_ao(ao_vector);
%          explore_ao(ao_matrix);
%
% HINT:    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
%
% VERSION: $Id: explore_ao.m,v 1.14 2008/02/14 10:41:02 ingo Exp $
%
% HISTORY: 10-06-07 Diepholz
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


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

```
%        Define the Positions         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% figure position
FIG_X   = .150;
FIG_Y   = .150;
FIG_dX  = .700;
FIG_dY = .600;

% tree position
TREE_X  = .000;
TREE_Y  = .000;
TREE_dX = .350;
TREE_dY = 1.000;

% plot field position
PLOT_X  = .370;
PLOT_Y  = .105;
PLOT_dX = .610;
PLOT_dY = .720;

% display field position
DISP_X  = TREE_X;
DISP_Y  = .750;
DISP_dX = TREE_dX;
DISP_dY = 1-DISP_Y;

% explorer name position
EXPL_NAME_dX = .370;
EXPL_NAME_dY = .045;
EXPL_NAME_X  = PLOT_X+(PLOT_dX-EXPL_NAME_dX)/2;
EXPL_NAME_Y  = .030;
% EXPL_NAME_Y  = .065;

% info fields position
N_INFOS = 3;

INFO_X  = PLOT_X;
INFO_Y  = .915;
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 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

        for gg = 1:n
          for hh = 1:m
            obj_value{end+1} = obj(gg,hh);
            obj_name{end+1}  = [ws_vars(ii).class ':' ws_vars(ii).name '(' num2str(gg) ',' num2str(hh) ')'];
          end
        end

      else
        error ('### this should not happen.');
      end

    end

  end
end


fig_name = ['explore the analysis object: ' tree_name];

% Define figure
fig = figure('Units',       'normalized',     ...
             'NextPlot',    'add',            ...
             'NumberTitle', 'off',            ...
             'Toolbar',     'none',           ...
             'name',         fig_name,        ...
             'Color',        BG_COLOR,        ...
             'Position',    [FIG_X  FIG_Y     ...
                             FIG_dX FIG_dY]);

%%%
```

```
root = ltpda_uitreenode(tree_name, tree_name, [], false);
tree = ltpda_uitree( fig,'Root',       root,             ...
                         'ExpandFcn', @myExpfcn4);
set(tree, 'Units',                 'normalized',         ...
          'position',               [TREE_X  TREE_Y      ...
                                     TREE_dX TREE_dY],   ...
          'NodeWillExpandCallback', @nodeWillExpand_cb4, ...
          'NodeSelectedCallback',   @nodeSelected_cb4);
tmp = tree.FigureComponent;
cell_Data = cell(3,1);

% cell_Data{1} = varargin{:};
cell_Data{1} = obj_value;
cell_Data{3} = obj_name;

set(tmp, 'UserData', cell_Data);


% Define the plot field
haxes = axes('Units',   'normalized',       ...
             'Position', [PLOT_X  PLOT_Y    ...
                          PLOT_dX PLOT_dY], ...
             'Box',     'on',               ...
             'XTick',    [],                ...
             'YTick',    []);

box  off;
axis off;

% Define the info fields 'name'
txt1 = uicontrol('String',          '',                  ...
                 'Units',           'normalized',        ...
                 'Style',           'Edit',              ...
                 'Position',         [INFO_X  INFO_Y     ...
                                      INFO_dX INFO_dY], ...
                 'BackgroundColor',   BG_COLOR);
% Define the info fields 'size'
txt2 = uicontrol('String',         '',                          ...
                 'Units',          'normalized',                ...
                 'Style',          'Edit',                      ...
                 'Position',        [(INFO_X+INFO_dX) INFO_Y    ...
                                      INFO_dX          INFO_dY],...
                 'BackgroundColor',   BG_COLOR);
% Define the info fields 'class'
txt3 = uicontrol('String',         '',                          ...
                 'Units',          'normalized',                ...
                 'Style',          'Edit',                      ...
                 '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
%%%%%%%%%%%%%%%%%%%
```

```
      if (isa(val,'history'))
        if length(val) == 1
          figure;
          plot (val);
          ii = strfind(plotted, 'inhists');
          title(haxes, sprintf('History-Level: %d', length(ii)+1))
        else
          na = text(0.5,0.5,'Select the left or right branch.');
          set(na, 'HorizontalAlignment', 'center', ...
            'Color',                'r',      ...
            'FontWeight',           'bold',   ...
            'EdgeColor',            'k',      ...
            'BackgroundColor',      'w',      ...
            'Fontsize',              10,      ...
            'Margin',                5);
        end

      %%%%%   Plot data object   %%%%%
      elseif isa(val,'fsdata') || isa(val,'tsdata') || ...
             isa(val,'xydata') || isa(val,'cdata')
        figure;
        plot(ao(val))

      %%%%%   Plot the AO object   %%%%%
      elseif isa(val, 'ao')
        figure;
        plot(ao(val))

      %%%%%   Plot mfir and miir object   %%%%%
      elseif isa(val, 'mfir') || isa(val, 'miir')
        figure;
        resp(val)

      %%%%%   Plot pzmodel object   %%%%%
      elseif isa(val, 'pzmodel')
        resp(val)

      %%%%%   Is the parent node == 'data' so plot data   %%%%%
      else

        cNode  = S{2};
        parent = '';

        if cNode.getLevel >= 1
          parent  = cNode.getParent;
        end

        if strcmp(parent.getValue, 'data')
          grandpa = parent.getParent;
          grandpa_val = grandpa.getValue;
          von = findstr(grandpa_val,'(');
          bis = findstr(grandpa_val,')');
          if ~isempty(von) && ~isempty(bis)
```

```
            index = grandpa_val(von+1:bis-1);
          else
            index = '1';
          end

          eval (sprintf('val = S{1}(%s);',index))
          figure;
          plot(val);
        end

      end

    end
%%%
```

```
(val,2))) )
      set(txt3, 'string', class(val))

      str = ' ';
      c_str = {};
      cla(haxes)

      if ~isempty(val)

        if isnumeric(val)
          %normalize the vector
          si = size(val);
          if si(1) > si(2)
            val = val.';
          else
            val = val;
          end
          if size(val,1) == 1 || size(val,2) == 1
            if length(val) > 3
              str = strcat(num2str(val(1:3)), '  ...');
            else
              str = num2str(val);
            end
          else
            str = 'Matrix';
          end

        elseif ischar(val)
          str = val;

        elseif islogical(val)
          if val
            str = 'true';
          else
            str = 'false';
          end

        elseif isobject(val)
          if isa(val, 'ao')
            str = 'Analysis Object';
          elseif isa(val, 'cdata')
            str = 'C-Data Object';
          elseif isa(val, 'fsdata')
            str = 'Frequency-Series Object';
          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')
            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)) = c_str1;
            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


%%%
```

```
      pth = '';
      eval (sprintf('pth = which(''%s'');',mfilename))
      index = find(pth==filesep, 1, 'last');
      pth = pth(1:index);
      pth = [pth 'exp_struct_icons' filesep];

      [n,m] = size(val);
      count = 0;

      %% Vector or Matrix
      if m>1 || n>1

        if isa(val, 'ao')
          iconpath =[pth,'analysis_object.gif'];
        else
          iconpath =[pth,'struct_icon.gif'];
        end

        %% Vector
        if m==1 || n==1
          L = length(val);

          for J = 1:L
            count = count + 1;
            cNode = S{2};

            level = cNode.getLevel;
            fname = strcat(cNode.getValue, '(', num2str(J),')');

            if level==0 && ~isempty(S{3}) && numel(S{3}) == numel(S{1})
              node_str = S{3}(J);
            else
              node_str = fname;
            end

            nodes(count) =  ltpda_uitreenode(fname, node_str, iconpath, 0);
          end
        %% Matrix
        else

          for ii=1:n
            for jj=1:m
              count = count + 1;
              cNode = S{2};
              fname = [cNode.getValue '(' num2str(ii) ',' num2str(jj) ')'];
              nodes(count) =  ltpda_uitreenode(fname, fname, iconpath, 0);
            end
          end

        end
      %% Struct, Object or single value
      else
        %%
```

```
        val = val;
        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

%%%
```

```
                  val = val{1};
                else
                  error('################ MAch mich neu');
                end
              else

                cmd = sprintf('val = getfield(val, field, {%s});',idx);
                eval(cmd);
                if iscell(val) && numel(val) == 1
                  val = val{1};
                end

              end

            else
              if iscell(val) && numel(val) == 1
                val = val{1};
              elseif numel(val) ~= 1
                error('################ MAch mich neu');
              end
              val = getfield(val, field);
            end
          end
        else
          displayed = cNode.getValue;
          if iscell(val) && numel(val) == 1
            val = val{1};
          else
          end
          return;
        end
    end


%%%
```

```

      c_str = strtrim(c_str);
    end

```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_obj_submit

PURPOSE ^

LTPDA_OBJ_SUBMIT submits the given collection of objects to an LTPDA Repository.

SYNOPSIS ^

function varargout = ltpda_obj_submit(varargin)

DESCRIPTION ^

```
 LTPDA_OBJ_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 ltpda_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] = ltpda_obj_submit(o1, sinfo)
              [ids, cids] = ltpda_obj_submit(o1,o2, sinfo)
              [ids, cids] = ltpda_obj_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
                 'username'               - the username for the server
                 'experiment_title'       - a title for the submission
                 'experiment_description' - a description of this submission
                 'reference_ids'          - a string containing any
                                            reference object id numbers
                 'additional_comments'    - any additional comments
                 'additional_authors'     - any additional author names

 VERSION:     $Id: ltpda_obj_submit.m,v 1.24 2008/02/24 19:24:31 hewitson Exp $


 HISTORY: 09-05-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- ltpda_insert LTPDA_INSERT inserts values into a single row of a table using JDBC driver
- mysql_getUserID MYSQL_GETUSERID gets the user ID number corresponding to the given user name.

This function is called by:

SUBFUNCTIONS ^

- function eid = update_object_tables(varargin)
- function id = getObjID(conn, hash)
- function Cid = getCollectionID(conn, objids)
- function s = csv(x)
- function check_sinfo(sinfo)
- function sinfo_description

SOURCE CODE ^

```
function varargout = ltpda_obj_submit(varargin)

% LTPDA_OBJ_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 ltpda_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] = ltpda_obj_submit(o1, sinfo)
%              [ids, cids] = ltpda_obj_submit(o1,o2, sinfo)
%              [ids, cids] = ltpda_obj_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
%                 'username'               - the username for the server
%                 'experiment_title'       - a title for the submission
%                 'experiment_description' - a description of this submission
%                 'reference_ids'          - a string containing any
%                                            reference object id numbers
```

```
%                  'additional_comments'    - any additional comments
%                  'additional_authors'     - any additional author names
%
% VERSION:     $Id: ltpda_obj_submit.m,v 1.24 2008/02/24 19:24:31 hewitson Exp $
%
%
% HISTORY: 09-05-07 M Hewitson
%             Creation
%
%%%
```

```
% Machine details
p = provenance;

%% Process each object and collect id numbers

disp('*** Submitting objects to repository...');

try

  % Look-up user id
  userid = mysql_getUserID(conn);
  disp(sprintf(' ** got user id %d for user: %s', userid, conn.Username))
  if userid < 1 || isnan(userid)  || strcmp(userid, 'No Data') || ischar(userid)
    error('### Unknown username.');
  end

  % The date for the transaction table
  t     = time();
  tdate = t.time_str;

  ids = [];
  for o=1:length(objs)
    % this object
    obj = objs{o};

    % Check this is a user object that can be submitted
    if ltpda_isuserobject(obj)

      % Object name
      if isfield(obj, 'name')
        name = obj.name;
      else
        name = '';
      end

      % Data object (if present)
      if isfield(obj, 'data') && ~isempty(obj.data)
        dname = obj.data.name;
        dtype = class(obj.data);
        ndata = length(obj.data.y);
      else
        dtype = '';
        dname = '';
        ndata = 0;
      end

      % Creation time
      if isfield(obj, 'created')
        if isa(obj.created, 'time')
          created = obj.created.time_str;
        elseif ischar(obj.created)
          created = obj.created;
        else
```

```
          error('### Unknown format for created field.');
        end
      else
        created = '1970-01-01 00:00:00';
      end
      if isempty(created)
        created = '1970-01-01 00:00:00';
      end

      % Version
      if isfield(obj, 'version')
        objver = obj.version;
      else
        objver = '';
      end

      %%%%%%%%%%% 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
      ltpda_xmlwrite(obj, xml, parent, '');    % Save the XML document.
      otxt = xmlwrite(xml); % then to string

      %%%%%%%%%%% Create MD5 hash of object
      md5hash = ltpda_hash(otxt, 'MD5');

      %%%%%%%%%%% Submit object to objs table

      try
        disp(sprintf('  + Uploading XML data...\f'));
        curs = exec(conn, ['insert into objs(xml,hash) values(''' otxt ''', '''
char(md5hash) ''' );'']);
        close(curs);
        disp(sprintf('\bdone.'));
      catch
        disp(sprintf('### Server returned: %s', curs.Message))
        error('### Submission error');
      end

      try
        objid = getObjID(conn, md5hash);
        disp(sprintf('  + submitted object %s with id %d', class(obj), objid));
      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 = ltpda_insert(conn, 'objmeta',...
          'obj_id', objid,...
          'obj_type', class(obj),...
          'name', name,...
          'created', created,...
          'version', objver,...
          'ip', p.ip,...
          'hostname', p.hostname,...
          'os', p.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...
          );

        disp(sprintf('  + made meta-data entry'));
      catch
        error('### failed to make meta-data entry');
      end

      %%%%%%%%%%% Update object tables
      eid = update_object_tables(conn, obj, objid);
      %%%%%%%%%%% Update transactions table

      try
        message = ltpda_insert(conn, 'transactions',...
          'obj_id', objid,...
          'user_id', userid,...
          'transdate', tdate,...
          'direction', 'in'...
          );
        disp(sprintf('  + updated transactions table'));
      catch
        error('### Failed to update transactions table');
      end

      % Collect this ID
      ids = [ids objid];
    else
      warning('!! Object is not one of the User classes. Not submitting.');
      disp(obj);
    end
```

```
  end

  %% Now make collection entry
  if ~isempty(ids)
    try
      message = ltpda_insert(conn, 'collections',...
        'nobjs', length(ids),...
        'obj_ids', csv(ids)...
        );
      disp(sprintf(' ** made collection entry'));
    catch
      error('### Failed to make an entry in the collections table');
    end
  end
catch
  disp('### Submission error - cleaning up.');
  rethrow(lasterror)
end

% Pass back outptus
varargout{1} = ids;
varargout{2} = getCollectionID(conn, ids);

disp('*** submission complete.');

end


%-----------------------------------------------------------------------
%
%  Update entries for various object tables
%
% eid = update_object_tables(conn, obj)
% eid = update_object_tables(conn, obj, objid)
%
function  eid = update_object_tables(varargin)

  if nargin < 2
    error('### Incorrect usage.');
  end
  conn = varargin{1};
  obj  = varargin{2};
  if nargin == 3
    objid = varargin{3};
  else
    objid = [];
  end

  % Class of object
  cl = class(obj);
  disp(sprintf('  + making meta data entry for %s', cl));
  eid = [];
```

```

  switch cl
    %------------------ AO
    case 'ao'

      %-- First we need to insert the data object info
      did = update_object_tables(conn, obj.data);

      % Now insert the AO info
      try
        message = ltpda_insert(conn, 'ao',...
          'obj_id', objid,...
          'data_type', class(obj.data),...
          'data_id', did, ...
          'description', obj.description, ...
          'mfilename', obj.mfilename, ...
          'mdlfilename', obj.mdlfilename...
          );
        disp(sprintf('  + made meta-data entry for %s', cl));
        % get the entry id
        eid = getObjID(conn, '');
      catch
        error('### failed to make meta-data entry for %s table', cl);
      end

    %------------------ CSDATA
    case 'cdata'
      % Make the meta-data entry
      try
        message = ltpda_insert(conn, 'cdata',...
          'xunits', obj.xunits,...
          'yunits', obj.yunits ...
          );
        disp(sprintf('  + made meta-data entry for %s', cl));
        % get the entry id
        eid = getObjID(conn, '');
      catch
        error('### failed to make meta-data entry for %s table', cl);
      end
    %------------------ FSDATA
    case 'fsdata'
      % Make the meta-data entry
      try
        message = ltpda_insert(conn, 'fsdata',...
          'xunits', obj.xunits,...
          'yunits', obj.yunits,...
          'fs', obj.fs ...
          );
        disp(sprintf('  + made meta-data entry for %s', cl));
        % get the entry id
        eid = getObjID(conn, '');
      catch
```

```
        error('### failed to make meta-data entry for %s table', cl);
      end
    %------------------ MFIR
    case 'mfir'
      % Make the meta-data entry
      try
        message = ltpda_insert(conn, 'mfir',...
          'obj_id', objid,...
          'in_file', obj.infile,...
          'fs', obj.fs ...
          );
        disp(sprintf('  + made meta-data entry for %s', cl));
        % get the entry id
        eid = getObjID(conn, '');
      catch
        error('### failed to make meta-data entry for %s table', cl);
      end
    %------------------ MIIR
    case 'miir'
      % Make the meta-data entry
      try
        message = ltpda_insert(conn, 'miir',...
          'obj_id', objid,...
          'in_file', obj.infile,...
          'fs', obj.fs ...
          );
        disp(sprintf('  + made meta-data entry for %s', cl));
        % get the entry id
        eid = getObjID(conn, '');
      catch
        error('### failed to make meta-data entry for %s table', cl);
      end
    %------------------ TSDATA
    case 'tsdata'
      % Make the meta-data entry
      try
        message = ltpda_insert(conn, 'tsdata',...
          'xunits', obj.xunits,...
          'yunits', obj.yunits,...
          'fs', obj.fs,...
          'nsecs', obj.nsecs,...
          't0', obj.t0.time_str ...
          );
        disp(sprintf('  + made meta-data entry for %s', cl));
        % get the entry id
        eid = getObjID(conn, '');
      catch
        error('### failed to make meta-data entry for %s table', cl);
      end
    %------------------ XYDATA
    case 'xydata'
      % Make the meta-data entry
```

```
      try
        message = ltpda_insert(conn, 'xydata',...
          'xunits', obj.xunits,...
          'yunits', obj.yunits ...
          );
        disp(sprintf('  + made meta-data entry for %s', cl));
        % get the entry id
        eid = getObjID(conn, '');
      catch
        error('### failed to make meta-data entry for %s table', cl);
      end
    otherwise
      disp(sprintf('   # no special table for objects of type %s', cl));
  end

  if ~isempty(eid)
    disp(sprintf('  + made meta-data entry in %s table with id %d', cl, eid));
  end
end

%-------------------------------------------------------------------------
%
function id = getObjID(conn, hash)

try
  q = sprintf('select last_insert_id()');
  curs = exec(conn, q);
  curs = fetch(curs);
  id  = curs.Data{1};
  close(curs);
catch
  error('### Failed to get object ID. Server returned %s', curs.Message);
end

end

%-------------------------------------------------------------------------
%
function Cid = getCollectionID(conn, objids)

try
  curs = exec(conn, sprintf('select id from collections where obj_ids="%s"', csv
(objids)));
  curs = fetch(curs);
  Cid  = curs.Data{1};
  close(curs);
catch
  error('### Failed to get Collection ID. Server returned %s', curs.Message);
end

if strcmp(Cid, 'No Data')
  Cid = [];
```

```
end

end

%-----------------------------------------------------------------------
% make comma separated list of numbers
function s = csv(x)

s = sprintf('%g,', x);
s = s(1:end-1);

end

%-----------------------------------------------------------------------
% check sinfo structure
%
function check_sinfo(sinfo)


% --- check fieldnames
manfields = {'conn', 'experiment_title', 'experiment_description', 'analysis_description'};
if ~isstruct(sinfo)
  sinfo_description();
end

% get input fieldnames
fnames = fieldnames(sinfo);

% check these
for jj=1:length(manfields)
  if ~ismember(fnames, manfields{jj})
    for kk=1:length(manfields)
      disp(sprintf('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

disp('*** 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-delimeted 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

% END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:03 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:03 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_phase

PURPOSE ^

LTPDA_PHASE return the phase in degrees for a given complex input.

SYNOPSIS ^

function p = ltpda_phase(resp)

DESCRIPTION ^

```
 LTPDA_PHASE return the phase in degrees for a given complex input.

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

 DESCRIPTION: LTPDA_PHASE return the phase in degrees for a given complex
              input. Supposed to be analogous to angle but return degrees
              instead of radians.

 CALL:        p = ltpda_phase(resp)

 INPUTS:      resp - complex number

 OUTPUTS:     p    - phase

 VERSION: $Id: ltpda_phase.m,v 1.5 2007/07/13 12:17:39 ingo Exp $

 HISTORY: dd-mm-yyyy M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- phase PHASE return the phase in degrees for a given complex input.

SOURCE CODE ^

```
0001 function p = ltpda_phase(resp)
```

```
% LTPDA_PHASE return the phase in degrees for a given complex input.
%
%%%
```

save

PURPOSE ^

SAVE a cdata object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE a cdata object to file.

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

 DESCRIPTION: SAVE overloads the save method for cdata objects.

 CALL:        Save a cdata object as a .mat file.
                >> save(d, 'blah.mat')
              Save a cdata object as an XML file.
                >> save(d, 'blah.xml')
              Save a cdata object to file specified in plist.
                >> save(d, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.4 2008/02/15 16:27:15 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(cdata, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(cdata,'Version')

 The following call returns a string that contains the routine category:

 >> category = save(cdata,'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a cdata object into a string.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE a cdata object to file.
%
%%%
```

```

ALGONAME = mfilename;
VERSION  = '$Id: save.m,v 1.4 2008/02/15 16:27:15 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'cdata') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

win_SFT5F

PURPOSE ^

WIN_SFT5F return SFT5F window sample for input z.

SYNOPSIS ^

function w = win_SFT5F(w, z)

DESCRIPTION ^

```
WIN_SFT5F return SFT5F window sample for input z.

M Hewitson 18-05-07

$Id: win_SFT5F.m,v 1.2 2008/02/13 18:38:03 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:03 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 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]

 GUI CALL:    The following call returns a parameter list object that contains
              the default parameter values:

              >> pl = ltpda_xcorr('Params')

 VERSION:     $Id: ltpda_polydetrend.m,v 1.8 2008/02/08 14:59:31 anneke Exp $

 HISTORY:     01-03-2007 M Hewitson
                 Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- polydetrend POLYDETREND detrends the input data vector with a polynomial.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = ltpda_polydetrend(varargin)
% LTPDA_POLYDETREND detrends the input analysis object using a polynomial of degree N.
%
%%%
```

```
  end
end

% capture input variable names
invars = {};
as     = [];
ps     = [];
for j=1:nargin
  invars = [invars cellstr(inputname(j))];
  if isa(varargin{j}, 'ao')
    as = [as varargin{j}];
  end
  if isa(varargin{j}, 'plist')
    ps = [ps varargin{j}];
  end
end

ALGONAME = mfilename;
VERSION  = '$Id: ltpda_polydetrend.m,v 1.8 2008/02/08 14:59:31 anneke Exp $';

% Look at inputs
if nargin < 1
  error('### Incorrect number of inputs.')
end

na = length(as);

% Combine plists
if ~isempty(ps)
  pl = combine(ps, getDefaultPL());
else
  pl = getDefaultPL();
end

% Initialise output
bo = [];

% Unpack parameter list
N = find(pl, 'N');

% Loop over analysis objects
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
```

```

  try
    if find(pl, 'M-FILE ONLY')
      y = polydetrend(d.y, N);
    else
      [y,coeffs] = ltpda_polyreg(d.y, N);
    end
  catch
    y = polydetrend(d.y, N);
  end

  % Make output analysis object
  nameStr = sprintf('poly_detrend(%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{i})
    n1 = a.name;
  else
    n1 = invars{i};
  end

  nameStr = sprintf('poly_detrend(%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();
plo = append(plo, param('N', 1));
```

```
disp('* done.');

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

save

PURPOSE ^

SAVE an fsdata object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE an fsdata object to file.

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

 DESCRIPTION: SAVE overloads the save method for fsdata objects.

 CALL:        Save an fsdata object as a .mat file.
                >> save(d, 'blah.mat')
              Save an fsdata object as an XML file.
                >> save(d, 'blah.xml')
              Save an fsdata object to file specified in plist.
                >> save(d, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.4 2008/02/15 16:27:15 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(fsdata, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(fsdata,'Version')

 The following call returns a string that contains the routine category:

 >> category = save(fsdata,'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE an fsdata object to file.
%
%%%
```

```

ALGONAME = mfilename;
VERSION  = '$Id: save.m,v 1.4 2008/02/15 16:27:15 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'fsdata') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

win_SFT5M

PURPOSE ^

WIN_SFT5M return SFT5M window sample for input z.

SYNOPSIS ^

function w = win_SFT5M(w, z)

DESCRIPTION ^

```
WIN_SFT5M return SFT5M window sample for input z.

M Hewitson 18-05-07

$Id: win_SFT5M.m,v 1.2 2008/02/13 18:38:03 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:03 ingo 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.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_polyreg Linux x86

PURPOSE ^

LTPDA_POLYREG detrends an input vector with a given order.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
LTPDA_POLYREG detrends an input vector with a given order.

M Hewitson 5-02-08

$Id: ltpda_polyreg.m,v 1.2 2008/02/21 14:34:55 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
% LTPDA_POLYREG detrends an input vector with a given order.
%
% M Hewitson 5-02-08
%
% $Id: ltpda_polyreg.m,v 1.2 2008/02/21 14:34:55 hewitson Exp $
%
```

save

PURPOSE ^

SAVE a history object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE a history object to file.

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

 DESCRIPTION: SAVE overloads the save method for history objects.

 CALL:        Save a history object as a .mat file.
                >> save(h, 'blah.mat')
              Save a history object as an XML file.
                >> save(h, 'blah.xml')
              Save a history object to file specified in plist.
                >> save(h, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.4 2008/02/15 16:27:15 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(history, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(history,'Version')

 The following call returns a string that contains the routine category:

 >> category = save(history,'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a param object into a string.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE a history object to file.
%
%%%
```

```

ALGONAME = mfilename;
VERSION  = '$Id: save.m,v 1.4 2008/02/15 16:27:15 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'history') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

Generated on Tue 26-Feb-2008 10:52:52 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 1.2 2008/02/13 18:38:03 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 1.2 2008/02/13 18:38:03 ingo Exp $
%

```

```
v = 1 - (2 * z - 1).^2;

% Make window struct
w.name     = '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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_psd

PURPOSE ^

LTPDA_PSD a psd estimation based on pwelch from MATLAB.

SYNOPSIS ^

function varargout = ltpda_psd(x, win, nolap, nfft, fs, scale, inunits, order)

DESCRIPTION ^

```
 LTPDA_PSD a psd estimation based on pwelch from MATLAB.

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

 DESCRIPTION: LTPDA_PSD a psd estimation based on pwelch from MATLAB.
              The difference is that detrending is done on each
              windowed data segment.

 CALL:       [f, pxx, info] = ltpda_psd(x, win, nolap, nfft, fs, scale, inunits)

 INPUTS:     x      - data vector
             win    - a window function
                      Specify a vector of window samples or a length. If a length
                      is specified, a Hamming window is used.
             nfft   - number of samples per fft
             nolap  - number of samples to overlap each segment by
             fs     - sample rate of input time-series
             scale  - specify one of 'ASD', 'AS', 'PSD', 'PS'
           inunits  - Amplitude units of input time-series data.
            order   - order of detrending

 OUTPUTS:    f      - a frequency vector
             Sxx    - spectral estimate in units specified by 'scale'
             info   - a structure of information

 NOTE:       segment_info() is copied directly from MATLAB's welshparse.m.

 VERSION:    $Id: ltpda_psd.m,v 1.4 2008/02/20 07:39:01 hewitson Exp $

 HISTORY:    25-09-2006 M Hewitson
                Creation

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = ltpda_psd('Params')

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpda_pwelch LTPDA_PWELCH makes power spectral density estimates of the time-series objects

SUBFUNCTIONS ^

- function [yy, info] = scale_xx(xx, win, fs, norm, inunits)
- function [L,noverlap,win,msg] = segment_info(M,win,noverlap)

SOURCE CODE ^

```
function varargout = ltpda_psd(x, win, nolap, nfft, fs, scale, inunits, order)
% LTPDA_PSD a psd estimation based on pwelch from MATLAB.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: LTPDA_PSD a psd estimation based on pwelch from MATLAB.
%              The difference is that detrending is done on each
%              windowed data segment.
%
% CALL:       [f, pxx, info] = ltpda_psd(x, win, nolap, nfft, fs,
scale, inunits)
%
% INPUTS:     x      - data vector
%             win    - a window function
%                      Specify a vector of window samples or a length. If
a length
%                      is specified, a Hamming window is used.
%             nfft   - number of samples per fft
%             nolap  - number of samples to overlap each segment by
%             fs     - sample rate of input time-series
%             scale  - specify one of 'ASD', 'AS', 'PSD', 'PS'
%           inunits  - Amplitude units of input time-series data.
%            order   - order of detrending
%
% OUTPUTS:    f      - a frequency vector
%             Sxx    - spectral estimate in units specified by 'scale'
%             info   - a structure of information
%
% NOTE:       segment_info() is copied directly from MATLAB's welshparse.m.
%
% VERSION:    $Id: ltpda_psd.m,v 1.4 2008/02/20 07:39:01 hewitson Exp $
%
% HISTORY:    25-09-2006 M Hewitson
%                Creation
%
% The following call returns a parameter list object that contains the
% default parameter values:
%
% >> pl = ltpda_psd('Params')
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% get segment information
Nx = length(x);
```

```
[L,noverlap,win,msg] = segment_info(Nx,win,nolap);
% Compute the number of segments
k = (Nx-noverlap)./(L-noverlap);

% Uncomment the following line to produce a warning each time the data
% segmentation does not produce an integer number of segements.
if fix(k) ~= k
   warning('The number of segments is not an integer, truncating data.');
end
k = fix(k);

% fix vectors
x   = x(:);
win = win(:);

% Initialise the spectral estimates
Sxx = zeros(nfft,1);

% loop over segments
LminusOverlap = L-noverlap;
xStart = 1:LminusOverlap:k*LminusOverlap;
xEnd   = xStart+L-1;

for i = 1:k,
  % data segment, detrend and window
  if order < 0
  else
    [y,coeffs] = ltpda_polyreg(x(xStart(i):xEnd(i)), order);
    dseg       = y.*win;
  end
  % compute fft
  xx   = fft(dseg,nfft);
  Sxx = ((Sxx .* (i-1)) +  xx.*conj(xx)) ./ i;
end
% take only one-sided
Sxx = Sxx(1:nfft/2+1);

% scale the data as required
[Sxx, info] = scale_xx(Sxx, win, fs, scale, inunits);

% grow frequency vector
f = linspace(0, fs/2, length(Sxx));

% complete info
info.nsecs = Nx/fs;
info.navs  = k;
info.nolap = noverlap/fs;

if nargout == 3

  varargout{1} = f;
  varargout{2} = Sxx;
  varargout{3} = info;


else
  disp('usage: [f, pxx, info] = ltpda_psd(x, win, nolap, nfft, fs,
scale, inunits, order)');
  error('# incorrect usage');
```

```
end



%-----------------------------------------------------------------------------------------------
function [yy, info] = scale_xx(xx, win, fs, norm, inunits)

nfft = length(win);
S1   = sum(win);
S2   = sum(win.^2);
enbw = fs * S2 / (S1*S1);

switch norm
  case 'ASD'
    yy = sqrt(xx) * sqrt(2 / (fs*S2));
    info.units = [sprintf('%s', inunits) '/\surdHz'];
  case 'PSD'
    yy = xx * 2 /(fs*S2);
    info.units = sprintf('%s^2/Hz', inunits);
  case 'AS'
    yy = sqrt(xx) * sqrt(2) / S1;
    info.units = sprintf('%s', inunits);
  case 'PS'
    yy = xx * 2 / (S1^2);
    info.units = sprintf('%s^2', inunits);
  case 'none'
    yy = xx;
    info.units = '';
  otherwise
    error('Unknown normalisation');
end

info.nfft = nfft;
info.enbw = enbw;
info.norm = norm;


%-----------------------------------------------------------------------------------------------
function [L,noverlap,win,msg] = segment_info(M,win,noverlap)
%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
%      MSG      - 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.

% Initialize outputs
L = [];
msg = '';

% Check that noverlap is a scalar
if any(size(noverlap) > 1),
    msg.identifier = generatemsgid('invalidNoverlap');
    msg.message = '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),
        msg.identifier = generatemsgid('invalidWindow');
        msg.message = '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,
    msg.identifier = generatemsgid('invalidSegmentLength');
    msg.message = 'The length of the segments cannot be greater than the
```

```
length of the input signal.';
    return;
end

if noverlap >= L,
    msg.identifier = generatemsgid('invalidNoverlap');
    msg.message = 'The number of samples to overlap must be less than
the length of the segments.';
    return;
end


%-----------------------------------------------------------------------------------------------------

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

save

PURPOSE ^

SAVE an mfir object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE an mfir object to file.

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

 DESCRIPTION: SAVE overloads the save method for mfir objects.

 CALL:        Save a mfir object as a .mat file.
                >> save(f, 'blah.mat')
              Save a mfir object as an XML file.
                >> save(f, 'blah.xml')
              Save a mfir object to file specified in plist.
                >> save(f, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.5 2008/02/15 16:27:16 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(mfir, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(mfir,'Version')

 The following call returns a string that contains the routine category:

 >> category = save(mfir,'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a mfir object into a string.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE an mfir object to file.
%
%%%
```

```

ALGONAME = mfilename;
VERSION  = '$Id: save.m,v 1.5 2008/02/15 16:27:16 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'mfir') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

wrapstring

PURPOSE ^

WRAPSTRING wraps a string to a cell array of strings with each cell less than n characters long.

SYNOPSIS ^

function s = wrapstring(s, n)

DESCRIPTION ^

```
 WRAPSTRING wraps a string to a cell array of strings with each cell less than n
characters long.

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

 DESCRIPTION: WRAPSTRING wraps a string to a cell array of strings with each
              cell less than n characters long.

 CALL:        s = wrapstring(s, n)

 INPUTS:      s  - String
              n  - max length of each cell

 OUTPUTS:     s  - the wrapped cell string

 VERSION: $Id: wrapstring.m,v 1.4 2007/07/13 12:17:39 ingo Exp $

 HISTORY: 05-02-2007 M Hewitson
              Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

- function s = splitstring(s,n)

SOURCE CODE ^

```
function s = wrapstring(s, n)
% WRAPSTRING wraps a string to a cell array of strings with each cell less than n characters long.
%
%%%
```

```
function s = splitstring(s,n)


% disp(sprintf('- splitting %s', s))
fi = 0;
idx = find(s==' ' | s == ',' | s == '(' | s == '=');
if max(idx) > n
  for i=1:length(idx)
    if idx(i) > n & fi==0
      fi = i;
    end
  end
  j = idx(fi);
  s = [cellstr(strtrim(s(1:j))) cellstr(strtrim(s(j+1:end)))];
else
  s = [];
  return;
end
```

Generated on Tue 26-Feb-2008 10:52:52 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 Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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

%%%
```

```
                Creation

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = ltpda_pwelch('Params')

 The following call returns a string that contains the routine CVS version:

 >> version = ltpda_pwelch(ao,'Version')

 The following call returns a string that contains the routine category:

 >> category = ltpda_pwelch(ao,'Category')

%%%
```

```
% 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: fs 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
%
% VERSION:    $Id: ltpda_pwelch.m,v 1.18 2008/02/20 07:39:01 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_pwelch('Params')
%
% The following call returns a string that contains the routine CVS version:
%
% >> version = ltpda_pwelch(ao,'Version')
%
% The following call returns a string that contains the routine category:
%
% >> category = ltpda_pwelch(ao,'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, pls, invars] = collect_inputs(varargin, in_names);
na = length(as);

% Loop over input AOs
for j=1:na

  % Unpack AO
  a = as(j);
  d = a.data;

  if isa(a, 'ao')
    % check input data
    if isa(d, 'tsdata')

      % combine plists
      plo = combine(pls, getDefaultPL());

      % check all defaults are sensible
      Nfft = find(plo, 'Nfft');   % Number of points in each fft
      setWindow = 0;
      if Nfft <= 0
        Nfft = length(d.y);
        disp(sprintf('! Using default Nfft of %g', Nfft))
        setWindow = 1;
      end
      plo = pset(plo, 'Nfft', Nfft);

      Win = find(plo, 'Win');   % Window object
      if setWindow || length(Win.win)~=Nfft
        switch Win.name
          case {'Kaiser', 'Flattop'}
            Win = specwin(Win.name, Nfft, Win.psll);
          otherwise
            Win = specwin(Win.name, Nfft);
        end
        disp(sprintf('! Reset window to %s(%d)', strrep(Win.name, '_', '\_'), length(Win.win)))
      end
      plo = pset(plo, 'Win', Win);

      olap = find(plo, 'olap');   % Amount to overlap each fft
      if olap <= 0
        olap = Win.rov;
```

```
        disp(sprintf('! Using default overlap of %d /%', olap))
      end
      plo = pset(plo, 'olap', olap);

      % Get detrending order
      order = find(plo, 'Order');

      % scaling of spectrum
      scale = find(plo, 'Scale');

      % check parameters
      if length(Win.win) ~= Nfft
        error('### Window length should be same as Nfft.');
      end

      % Compute PSD using pwelch
      [f, pxx, info] = ltpda_psd(d.y, Win.win, round(olap/100*Nfft), Nfft, d.fs, scale, d.yunits, order);

      % create new output fsdata
      fs = fsdata(f, pxx, d.fs);
      fs = set(fs, 'name', sprintf('%s(%s)', scale, d.name));
      fs = set(fs, 'yunits', info.units);
      fs = set(fs, 'enbw', Win.nenbw*d.fs/Nfft);
      fs = set(fs, 'navs', info.navs);

      % create new output history
      h = history(ALGONAME, VERSION, plo, a.hist);
      h = set(h, 'invars', invars);

      % make output analysis object
      b = ao(fs, h);

      % set name
      if na~=length(invars) || isempty(invars{j})
        nj = a.name;
      else
        nj = invars{j};
      end
      b = setnh(b, 'name', sprintf('%s(%s)', scale, nj));

      % add to outputs
      bs = [bs b];

    else
      warning('### Ignoring input AO number %d (%s); it is not a time-series.', j, a.name)
    end
  end
end % loop over analysis objects

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

```


%-----------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('Nfft', -1));
plo = append(plo, param('Win', specwin('Kaiser', 1, 200)));
plo = append(plo, param('Olap', -1));
plo = append(plo, param('Scale', 'PSD'));
plo = append(plo, param('Order', 0));


```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

save

PURPOSE ^

SAVE an miir object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE an miir object to file.

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

 DESCRIPTION: SAVE overloads the save method for miir objects.

 CALL:        Save a miir object as a .mat file.
                >> save(f, 'blah.mat')
              Save a miir object as an XML file.
                >> save(f, 'blah.xml')
              Save a miir object to file specified in plist.
                >> save(f, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.4 2008/02/15 16:27:16 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(miir, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(miir,'Version')

 The following call returns a string that contains the routine category:

 >> category = save(miir,'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a miir object into a string.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE an miir object to file.
%
%%%
```

```

ALGONAME = mfilename;
VERSION  = '$Id: save.m,v 1.4 2008/02/15 16:27:16 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'miir') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

xdoc2ao

PURPOSE ^

XDOC2AO converts a DOM object to an AO.

SYNOPSIS ^

function aout = xdoc2ao(xdoc)

DESCRIPTION ^

```
XDOC2AO converts a DOM object to an AO.

M Hewitson 16-05-07

$Id: xdoc2ao.m,v 1.2 2007/11/14 16:30:18 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function aout = xdoc2ao(xdoc)

% XDOC2AO converts a DOM object to an AO.
%
% M Hewitson 16-05-07
%
% $Id: xdoc2ao.m,v 1.2 2007/11/14 16:30:18 ingo Exp $
%

```

```
ain = xdoc.getElementsByTagName('a');
a = ain.item(0);

children = a.getChildNodes;

name       = '';
prov       = [];
description= '';
mfile      = '';
mfname     = '';
mdlfile    = '';
mdlfname   = '';
hist       = [];
data       = [];

% Go through children of AO object
for j=1:children.getLength

  ch = children.item(j-1);

  nodeName = char(ch.getNodeName);
  if ischar(nodeName)
    switch nodeName
      case 'Name'
        name = char(ch.getTextContent);
      case 'Provenance'
        prov = xml_read_provenance(ch);
      case 'Description'
        description = char(ch.getTextContent);
      case 'M-file'
        mfile = char(ch.getTextContent);
      case 'Mfilename'
        mfname = char(ch.getTextContent);
      case 'Mdl-file'
        mdlfile = char(ch.getTextContent);
      case 'Mdlfilename'
        mdlfname = char(ch.getTextContent);
      case 'hist'
        hist = xml_read_hist(ch);
      case 'Data'
        data = xml_read_data(ch);
      otherwise
%         warning(['!!! unknown child node: ' nodeName]);
```

```
    end
  end
end

% Now build AO
aout.name        = name;
aout.data        = data;
aout.hist        = hist;
aout.provenance  = prov;
aout.description = description;
aout.mfile       = mfile;
aout.mfilename   = mfname;
aout.mdlfile     = mdlfile;
aout.mdlfilename = mdlfname;
aout = class(aout, 'ao');


```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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 2007/11/11 16:50:38 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

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 2007/11/11 16:50:38 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
```

```
0052
0053   n = n + 1;
0054   lbuffer = fgets(in);
0055 end
0056 fclose(in);
0057
0058 if(nacoeffs > nbcoeffs)
0059   error('## unstable filter: nacoeffs > nbcoeffs');
0060 end
0061
0062 if(nacoeffs < nbcoeffs)
0063   filt.a = [filt.a zeros(1,(nbcoeffs-nacoeffs))];
0064 end
0065
0066 ncoeffs = nbcoeffs;
0067
0068
0069 f.name    = name;
0070 f.fs      = fs;
0071 f.ntaps   = ncoeffs;
0072 f.a       = filt.a;
0073 f.b       = filt.b;
0074 f.gain    = 1;
0075 f.histin  = zeros(1, f.ntaps-1);
0076 f.histout = zeros(1, f.ntaps-1);
0077
0078
0079
0080 % END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_rand

PURPOSE ^

LTPDA_RAND return a random number between r1 and r2

SYNOPSIS ^

function val = ltpda_rand(r1, r2)

DESCRIPTION ^

```
 LTPDA_RAND return a random number between r1 and r2

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

 DESCRIPTION: LTPDA_RAND return a random number between r1 and r2

 CALL:       val = ltpda_rand(r1, r2)

 INPUTS:     r1  - lower limit of the random values
             r2  - upper limit of the random values

 OUTPUTS:    val - random numbers

 VERSION: $Id: ltpda_rand.m,v 1.2 2007/07/13 12:17:39 ingo Exp $

 HISTORY: 26-01-2007 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function val = ltpda_rand(r1, r2)
% LTPDA_RAND return a random number between r1 and r2
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%
% DESCRIPTION: LTPDA_RAND return a random number between r1 and r2
%
% CALL:       val = ltpda_rand(r1, r2)
%
% INPUTS:     r1  - lower limit of the random values
%             r2  - upper limit of the random values
%
% OUTPUTS:    val - random numbers
%
% VERSION: $Id: ltpda_rand.m,v 1.2 2007/07/13 12:17:39 ingo Exp $
%
% HISTORY: 26-01-2007 M Hewitson
%             Creation
%
%%%
```

xydata

PURPOSE ^

XYDATA X-Y data object class constructor.

SYNOPSIS ^

function xy = xydata(varargin)

DESCRIPTION ^

```
 XYDATA X-Y data object class constructor.

%%%
```

- subsasgn SUBSASGN define index assignment for xydata properties.

SUBFUNCTIONS ^

- function xy = init()

SOURCE CODE ^

```
function xy = xydata(varargin)
% XYDATA X-Y data object class constructor.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: XYDATA X-Y data object class constructor.
%              Create an X-Y data object.
%
%     Properties:
%       name    - name of time-series object
%       x       - x samples vector
%       y       - y samples vector
%       xunits  - units to interpret the time samples (e.g., seconds)
%       yunits  - units to interpret the data samples (e.g., Volts)
%       version - current version of the constructor
%       created - current date and time as an number
%
%     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.
%
% HISTORY: 30-01-2007 Hewitson
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: xydata.m,v 1.14 2008/02/13 12:58:11 mauro Exp $';
CATEGORY = 'Constructor';

%%%%%   Is this a 'Params' call?     %%%%%
if  nargin == 2 && ischar(varargin{2})
  if strcmp(varargin{2}, 'Params')
    xy = plist();
    return
  elseif strcmp(varargin{2}, 'Version')
    xy = VERSION;
    return
  elseif strcmp(varargin{2}, 'Category')
    xy = CATEGORY;
    return
```

```
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%   define xydata properties   %%%%%%%%%%%%%%%%%%%%%%%%

  function xy = init()
    xy.name    = 'None';
    xy.x       = [];
    xy.y       = [];
    xy.xunits  = '';
    xy.yunits  = '';
    xy.version = VERSION;
    xy.created = time;
    xy         = class(xy, 'xydata');
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%   Create xydata object   %%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%   xy = xydata()   %%%%%%%%%%
% create default xydata object
if nargin == 0
  xy = init();

elseif nargin == 1

  %%%%%%%%%%% From File %%%%%%%%%%%%%%%%
  if ischar(varargin{1})

    filename = varargin{1};
    [path, name, ext, vers] = fileparts(filename);
    switch ext
      case '.mat'
        xy = load(filename);
        xy = xy.a;
      case '.xml'
        root_node = xmlread(filename);
        xy = ltpda_xmlread(root_node, 'xydata');
      otherwise
        error('### Unknown file type.');
    end
  %%%%%%%%%%   xy = xydata(xydata)   %%%%%%%%%%
  elseif isa(varargin{1}, 'xydata')
    xy = varargin{1};

  %%%%%%%%%%   xy = xydata(struct)   %%%%%%%%%%
  elseif isstruct(varargin{1})

    xy = init();

    fields = fieldnames(varargin{1});
    for ii = 1:length(fields)
      field = fields{ii};
```

```
      %%% created -> time-object
      if strcmp(field, 'created')
        created       = varargin{1}.created;
        if isstruct(created)
          created = time(created);
        end
        xy.created     = created;
      %%% All other
      else
        try
          xy.(field) = varargin{1}.(field);
        catch
          error('### The field ''%s'' in the struct is not a xydata property.', field)
        end
      end
    end

  %%%%%%%%%%   xy = xydata(y_vector)   %%%%%%%%%%
  else
    xy   = init();
    xy.x = 1:length(varargin{1});
    xy.y = varargin{1};
  end

%%%%%%%%%%   xy = xydata(x_vector, y_vector)   %%%%%%%%%%
elseif nargin == 2

  %%%%%%%%%%% From DATABASE
  if isa(varargin{1}, 'database')
    xy = retrieve(varargin{1}, varargin{2:end});
    return
  else
    xy   = init();
    xy.x = varargin{1};
    xy.y = varargin{2};
  end
else
  error('### Unknown number of constructor arguments.');
end


%%%%%%%%%%   Normalize the the x- and y-vector   %%%%%%%%%%
if size(xy.y,2) > size(xy.y,1)
  xy.y = [xy.y].';
end
xy.x = reshape(xy.x, size(xy.y));


end % function xy = xydata(varargin)


```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_reformat_date

PURPOSE ^

LTPDA_REFORMAT_DATE reformats the input date

SYNOPSIS ^

function s = ltpda_reformat_date (s)

DESCRIPTION ^

```
 LTPDA_REFORMAT_DATE reformats the input date

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

 DESCRIPTION: LTPDA_REFORMAT_DATE reformats the input date of the (preferred)
              form 'yyyy-mm-dd HH:MM:SS' to the format 'mm-dd-yyyy HH:MM:SS'
              which works with datenum.

 CALL:        s  = ltpda_reformat_date(si)

 INPUTS:      si - date string in the format 'yyyy-mm-dd HH:MM:SS'

 OUTPUTS:     s  - date string in the format 'mm-dd-yyyy HH:MM:SS'

 EXAMPLE:     dnum = datenum(ltpda_reformat_date('2007-05-04 22:34:22'));

 VERSION: $Id: ltpda_reformat_date.m,v 1.1 2007/07/25 09:21:35 ingo Exp $

 HISTORY: 03-06-2007 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function s = ltpda_reformat_date (s)
```

```
% LTPDA_REFORMAT_DATE reformats the input date
%
%%%
```

xyzdata

PURPOSE ^

XZYDATA X-Y-Z data object class constructor.

SYNOPSIS ^

function xyz = xyzdata(varargin)

DESCRIPTION ^

```
 XZYDATA X-Y-Z data object class constructor.

%%%
```

This function is called by:

- subsasgn SUBSASGN define index assignment for xyzdata properties.

SUBFUNCTIONS ^

- function xyz = init()

SOURCE CODE ^

```
function xyz = xyzdata(varargin)
% XZYDATA X-Y-Z data object class constructor.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: XYZDATA X-Y-Z data object class constructor.
%              Create an X-Y-Z data object.
%
%     Properties:
%       name    - name of time-series object
%       x       - x samples vector
%       y       - y samples vector
%       z       - z samples vector
%       xunits  - units to interpret the time samples (e.g., seconds)
%       yunits  - units to interpret the data samples (e.g., frequency)
%       zunits  - units to interpret the data samples (e.g., Volts)
%       version - current version of the constructor
%       created - current date and time as an number
%
%     Possible constructors:
%       xyz = xyzdata()      - creates a blank xyz object
%       xyz = xyzdata(y)     - 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.
%
% HISTORY: 24-12-2007 Hewitson
%             Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VERSION  = '$Id: xyzdata.m,v 1.3 2008/02/13 13:22:45 mauro Exp $';
CATEGORY = 'Constructor';

%%%%%   Is this a 'Params' call?     %%%%%
if  nargin == 2 && isa(varargin{1}, 'xyzdata') && ischar(varargin{2})
  if strcmp(varargin{2}, 'Params')
    xyz = plist();
    return
  elseif strcmp(varargin{2}, 'Version')
    xyz = VERSION;
```

```
    return
  elseif strcmp(varargin{2}, 'Category')
    xyz = CATEGORY;
    return
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%   define xydata properties   %%%%%%%%%%%%%%%%%%%%%%%%

  function xyz = init()
    xyz.name    = 'None';
    xyz.x       = [];
    xyz.y       = [];
    xyz.z       = [];
    xyz.xunits  = '';
    xyz.yunits  = '';
    xyz.zunits  = '';
    xyz.version = VERSION;
    xyz.created = time;
    xyz         = class(xyz, 'xyzdata');
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%   Create xydata object   %%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%   xy = xydata()   %%%%%%%%%%
% create default xydata object
if nargin == 0
  xyz = init();

elseif nargin == 1

  %%%%%%%%%%%% From File %%%%%%%%%%%%%%%%
  if ischar(varargin{1})

    filename = varargin{1};
    [path, name, ext, vers] = fileparts(filename);
    switch ext
      case '.mat'
        xyz = load(filename);
        xyz = xyz.a;
      case '.xml'
        root_node = xmlread(filename);
        xyz = ltpda_xmlread(root_node, 'xyzdata');
      otherwise
        error('### Unknown file type.');
    end
  %%%%%%%%%%   xy = xydata(xydata)   %%%%%%%%%%
  elseif isa(varargin{1}, 'xyzdata')
    xyz = varargin{1};

  %%%%%%%%%%   xy = xydata(struct)   %%%%%%%%%%
  elseif isstruct(varargin{1})

```

```
    xyz = init();

    fields = fieldnames(varargin{1});
    for ii = 1:length(fields)
      field = fields{ii};

      %%% created -> time-object
      if strcmp(field, 'created')
        created       = varargin{1}.created;
        if isstruct(created)
          created = time(created);
        end
        xyz.created     = created;
      %%% All other
      else
        try
          xyz.(field) = varargin{1}.(field);
        catch
          error('### The field ''%s'' in the struct is not a xydata property.', field)
        end
      end
    end

  %%%%%%%%%%   xyz = xyzdata(y_vector)   %%%%%%%%%%
  else
    xyz   = init();
    xyz.x = 1:size(varargin{1}, 1);
    xyz.y = 1:size(varargin{1}, 2);
    xyz.z = varargin{1};
  end

%%%%%%%%%%   xy = xydata(x_vector, y_vector)   %%%%%%%%%%
elseif nargin == 2

  %%%%%%%%%%% From DATABASE
  if isa(varargin{1}, 'database')
    xyz = retrieve(varargin{1}, varargin{2:end});
  else
    error('### Unknown constructor.');
  end
elseif nargin == 3
  xyz   = init();
  xyz.x = varargin{1};
  xyz.y = varargin{2};
  xyz.z = varargin{3};
else
  error('### Unknown number of constructor arguments.');
end


%%%%%%%%%%   Normalize the the x- and y-vector   %%%%%%%%%%
```

```

nx = size(xyz.z, 2);
ny = size(xyz.z, 1);

if length(xyz.x) ~= nx
  error('### X-vector has wrong length.');
end
if length(xyz.y) ~= ny
  error('### Y-vector has wrong length.');
end
if size(xyz.y,2) > size(xyz.y,1)
  xyz.y = [xyz.y].';
end
if size(xyz.x,2) > size(xyz.x,1)
  xyz.x = [xyz.x].';
end



end % function xy = xydata(varargin)


```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

ltpda_ri2fq

PURPOSE ^

LTPDA_RI2FQ Convert complex pole/zero into frequency/Q pole/zero representation.

SYNOPSIS ^

function [f0, q]= ltpda_ri2fq(c)

DESCRIPTION ^

```
LTPDA_RI2FQ Convert complex pole/zero into frequency/Q pole/zero representation.

 [f0, q]= ltpda_ri2fq(c)

M Hewitson 26-01-07

$Id: ltpda_ri2fq.m,v 1.4 2007/01/26 14:03:24 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function [f0, q]= ltpda_ri2fq(c)

% LTPDA_RI2FQ Convert complex pole/zero into frequency/Q pole/zero representation.
%
%  [f0, q]= ltpda_ri2fq(c)
%
% M Hewitson 26-01-07
%
% $Id: ltpda_ri2fq.m,v 1.4 2007/01/26 14:03:24 hewitson Exp $
%


if(nargin==0)
```

```
  disp('usage: [f0, q]= ltpda_ri2fq(c)');
  return
end

a = real(c);
b = imag(c);

f0 = -a*sqrt(b^2/a^2 + 1) / 2 / pi;
q = sqrt(b^2/a^2 +1)/2;

% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

save

PURPOSE ^

SAVE a pole object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE a pole object to file.

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

 DESCRIPTION: SAVE overloads the save method for pole objects.

 CALL:        Save a pole object as a .mat file.
                >> save(p, 'blah.mat')
              Save a pole object as an XML file.
                >> save(p, 'blah.xml')
              Save a pole object to file specified in plist.
                >> save(p, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.4 2008/02/15 16:27:16 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(pole, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(pole,'Version')

 The following call returns a string that contains the routine category:

 >> category = save(pole,'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a pole object into a string.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE a pole object to file.
%
%%%
```

```

ALGONAME = mfilename;
VERSION  = '$Id: save.m,v 1.4 2008/02/15 16:27:16 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'pole') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

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

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = ltpda_rms('Params')

 VERSION:     $Id: ltpda_rms.m,v 1.8 2007/11/08 10:35:04 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:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = ltpda_rms(varargin)
% LTPDA_RMS Calculate RMS deviation from spectrum
%
%%%
```

```
spe = [d.x d.y];
f   = d.x;

% 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

% r=[freq,sqrt(ms)];

% Create output analysis object
% create new output fsdata
fs = fsdata(freq, sqrt(ms), d.fs);
fs = set(fs, 'name', sprintf('RMS(%s)', d.name));
fs = set(fs, 'yunits', d.yunits);

% create new output history
h = history(ALGONAME, VERSION, [], a.hist);

% make output analysis object
b = ao(fs, h);

% set name
b = setnh(b, 'name', sprintf('RMS(%s)', a.name));

varargout{1} = b;


%-------------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

disp('* creating default plist...');
plo = plist();
disp('* done.');
```

```
0100
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003

save

PURPOSE ^

SAVE a provenance object to file.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE a provenance object to file.

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

 DESCRIPTION: SAVE overloads the save method for provenance objects.

 CALL:        Save a provenance object as a .mat file.
                >> save(p, 'blah.mat')
              Save a provenance object as an XML file.
                >> save(p, 'blah.xml')
              Save a provenance object to file specified in plist.
                >> save(p, plist)

 Parameters: 'filename' - name of the file

 VERSION:     $Id: save.m,v 1.3 2008/02/13 17:11:03 mauro Exp $

 The following call returns a parameter list object that contains the
 default parameter values:

 >> pl = save(provenance, 'Params')

 The following call returns a string that contains the routine CVS version:

 >> version = save(provenance, 'Version')

 The following call returns a string that contains the routine category:

 >> category = save(provenance, 'Category')

 HISTORY: 28-08-07 M Hewitson
             Creation

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

CROSS-REFERENCE INFORMATION ^

This function calls:

- char CHAR convert a provenance object into a string.

This function is called by:

SUBFUNCTIONS ^

- function plo = getDefaultPL()

SOURCE CODE ^

```
function varargout = save(varargin)
% SAVE a provenance object to file.
%
%%%
```

```

VERSION  = '$Id: save.m,v 1.3 2008/02/13 17:11:03 mauro Exp $';
CATEGORY = 'Output';

% Check if this is a call for parameters
if nargin == 2
  if isa(varargin{1}, 'provenance') && ischar(varargin{2})
    in = char(varargin{2});
    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
end

if nargin ~= 2
  help mfilename;
  error('### incorrect inputs');
end

% get object
obj = varargin{1};
if isa(varargin{2}, 'plist')
  pl = varargin{2};
elseif ischar(varargin{2})
  pl = plist(param('filename', varargin{2}));
else
  help mfilename;
  error('### incorrect inputs');
end

% Save object
ltpda_saveobj(obj, pl);

% Get default params
function plo = getDefaultPL()

plo = plist();
plo = append(plo, param('filename', 'object.xml'));


% END
```

Generated on Tue 26-Feb-2008 10:52:52 by **m2html** © 2003