



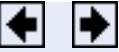
LTPDA Toolbox

0.0036 0.0036
0.0036 0.0036 0.0036
0.0046 0.0046

- [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.

[!\[\]\(cbe2492b119e39e02a1dab2af4a4b296_img.jpg\) Functions - Alphabetical List](#)[Getting Started with the LTPDA Toolbox !\[\]\(e474458956c9a37fbf9586ddb60a7fa1_img.jpg\)](#)

©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	



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.

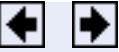
Component	Version	Comment
MATLAB	7.5 (R2007b)	
Simulink	7.0	Needed for GUI programming

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.

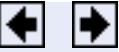
Automatically start LTPDA Toolbox on starting MATLAB

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



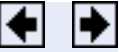
Starting the LTPDA Toolbox

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



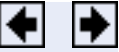
Examples

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

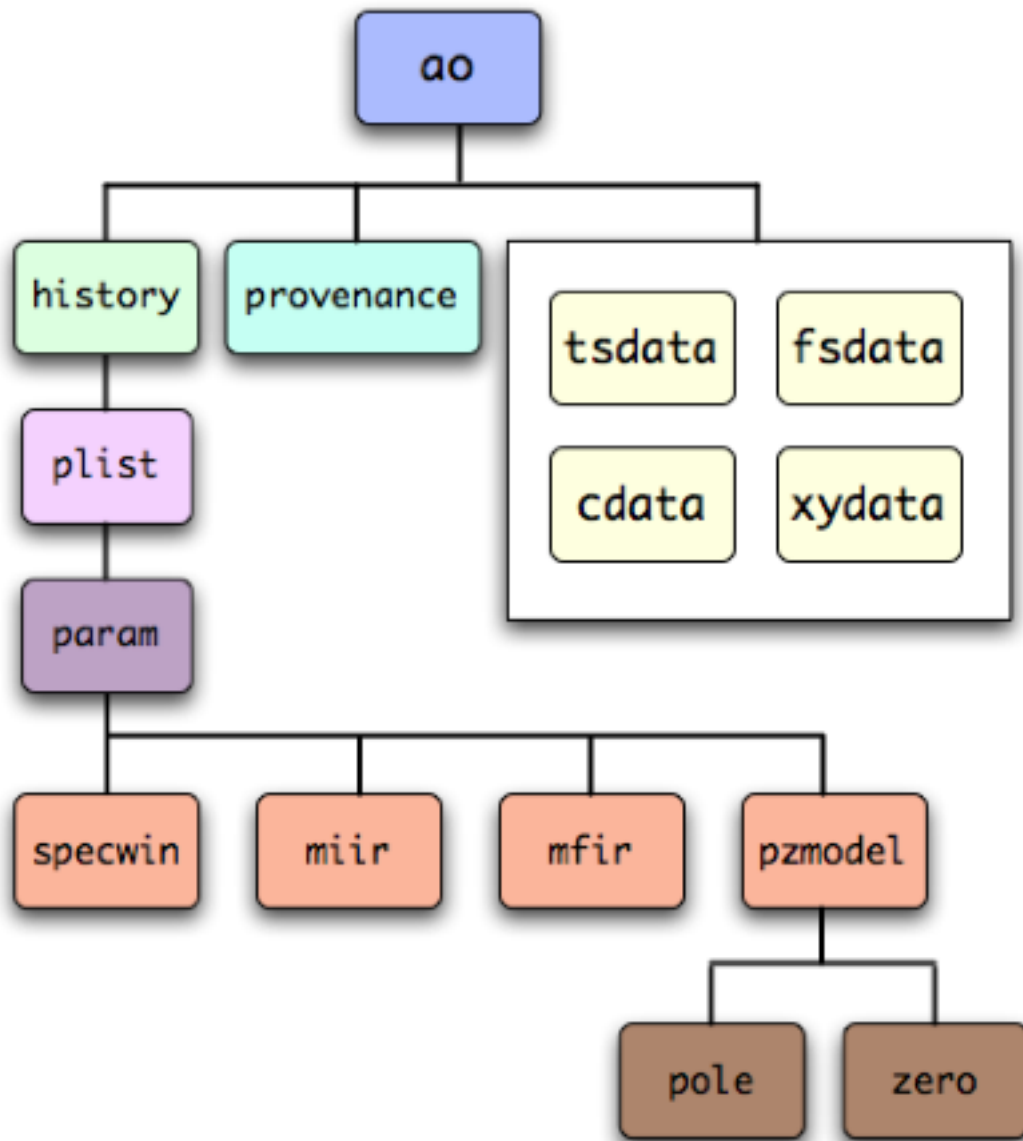
- which raw data was involved (date, channel, time segment, time of retrieval if data can be changed later by new downlinks)
- all operations performed on the data
- the above for all channels of a multi-channel plot

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 1.6 2007/11/14
11:20:09 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 1.6 2007/11/14
11:20:09 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 1.6 2007/11/14
11:20:09 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 1.6 2007/11/14
11:20:09 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 1.6 2007/11/14
11:20:09 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 1.6 2007/11/14
```

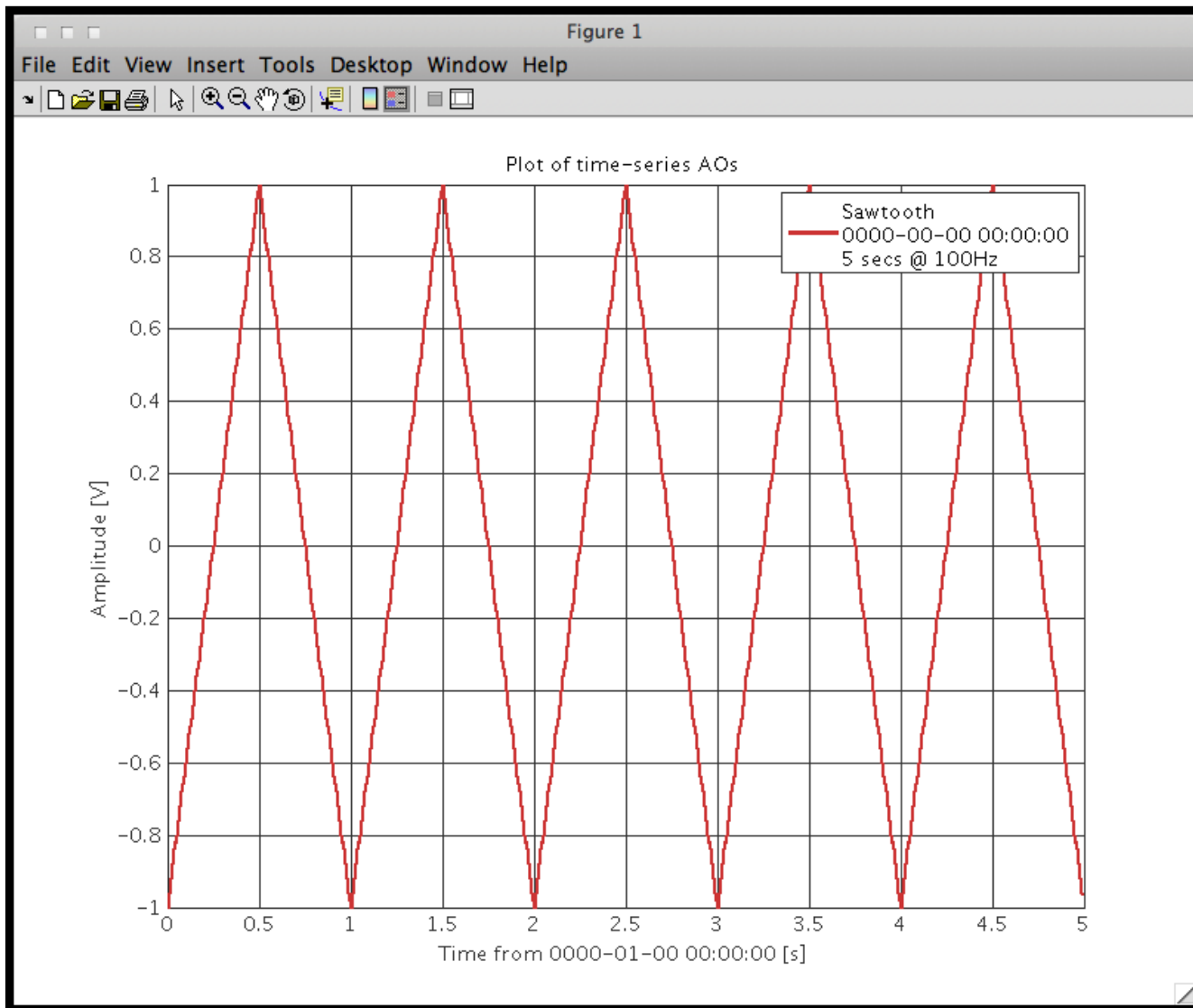
```
11:20:09 hewitson Exp $
```

```
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

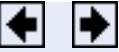
```

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.



Saving Analysis Objects

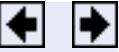
Analysis Objects can be saved to disk as either MATLAB binary files (.MAT) or as XML files (.XML). The following code shows how to do this:

```
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](#)

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 parameter object can be accessed as follows:

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



Creating lists of Parameters

Parameters can be grouped together into parameter lists (`plist`). The following code shows how to create a parameter list from individual parameters.

```

>> p1 = param('a', 1); % create first parameter
>> p2 = 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
-----
```

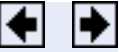


Creating Parameters

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](#)



What are LTPDA spectral windows?

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

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

Window name	NENBW	PSLL [dB]	ROV [%]
Rectangular	1.000	-13.3	0.0
Welch	1.200	-21.3	29.3
Bartlett	1.333	-26.5	50.0
Hanning	1.500	-31.5	50.0
Hamming	1.363	-42.7	50.0
Nuttall3	1.944	-46.7	64.7
Nuttall4	2.310	-60.9	70.5
Nuttall3a	1.772	-64.2	61.2

Nuttall3b	1.704	-71.5	59.8
Nuttall4a	2.125	-82.6	68.0
Nuttall4b	2.021	-93.3	66.3
Nuttall4c	1.976	-98.1	65.6
BH92	2.004	-92.0	66.1
SFT3F	3.168	-31.7	66.7
SFT3M	2.945	-44.2	65.5
FTNI	2.966	-44.4	65.6
SFT4F	3.797	-44.7	75.0
SFT5F	4.341	-57.3	78.5
SFT4M	3.387	-66.5	72.1
FTHP	3.428	-70.4	72.3
HFT70	3.413	-70.4	72.2
FTSRS	3.770	-76.6	75.4
SFT5M	3.885	-89.9	76.0

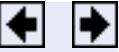
HFT90D	3.883	-90.2	76.0
HFT95	3.811	-95.0	75.6
HFT116D	4.219	-116.8	78.2
HFT144D	4.539	-114.1	79.9
HFT169D	4.835	-169.5	81.2
HFT196D	5.113	-196.2	82.3
HFT223D	5.389	-223.0	83.3
HFT248D	5.651	-248.0	84.1

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

◀ Spectral Windows

Create spectral windows ▶

©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, PSL, 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 

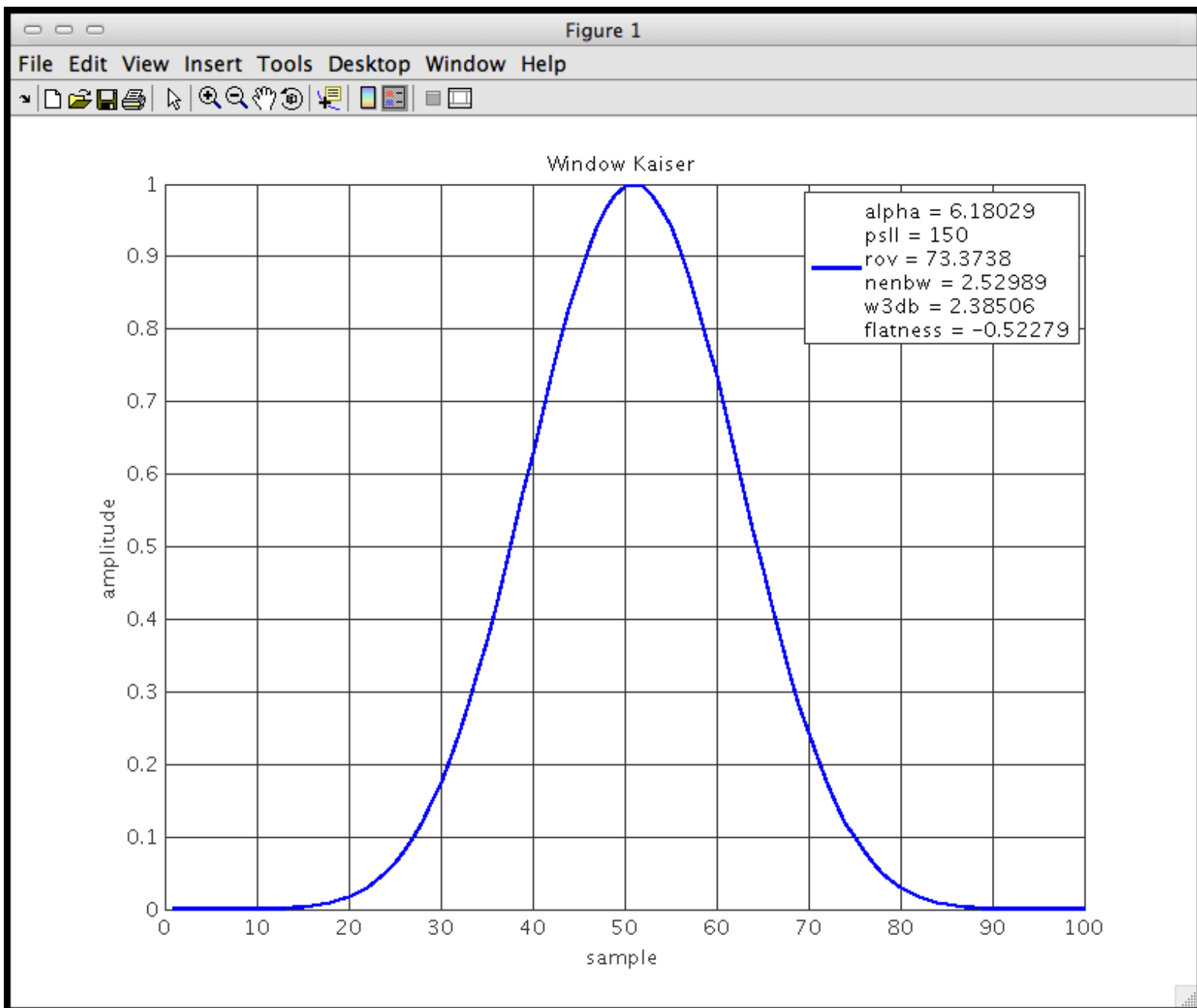
©LTP Team



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);
```

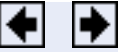
In this case, the size of the spectral window (number of samples) may not match the length of the segments in the spectral estimation. The `ltpda_pwelch` algorithm then recomputes the window using the input design but for the correct length of window function.

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`.

©LTP Team



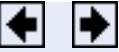
Simulation/modelling

Content needs written...

 Using spectral windows

Generating model noise 

©LTP Team



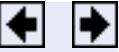
Generating model noise

Content needs written...

 Simulation/modelling

Frankin noise-generator 

©LTP Team



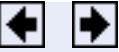
Franklin noise-generator

Discussion about Franklin's algorithm.

 [Generating model noise](#)

[Pole/Zero Modelling](#) 

©LTP Team



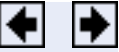
Pole/Zero Modelling

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](#)





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.

Building a model

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

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

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

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

Direct form

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

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

Creating from a plist

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

```
>> pl = plist();
>> 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)]));
```

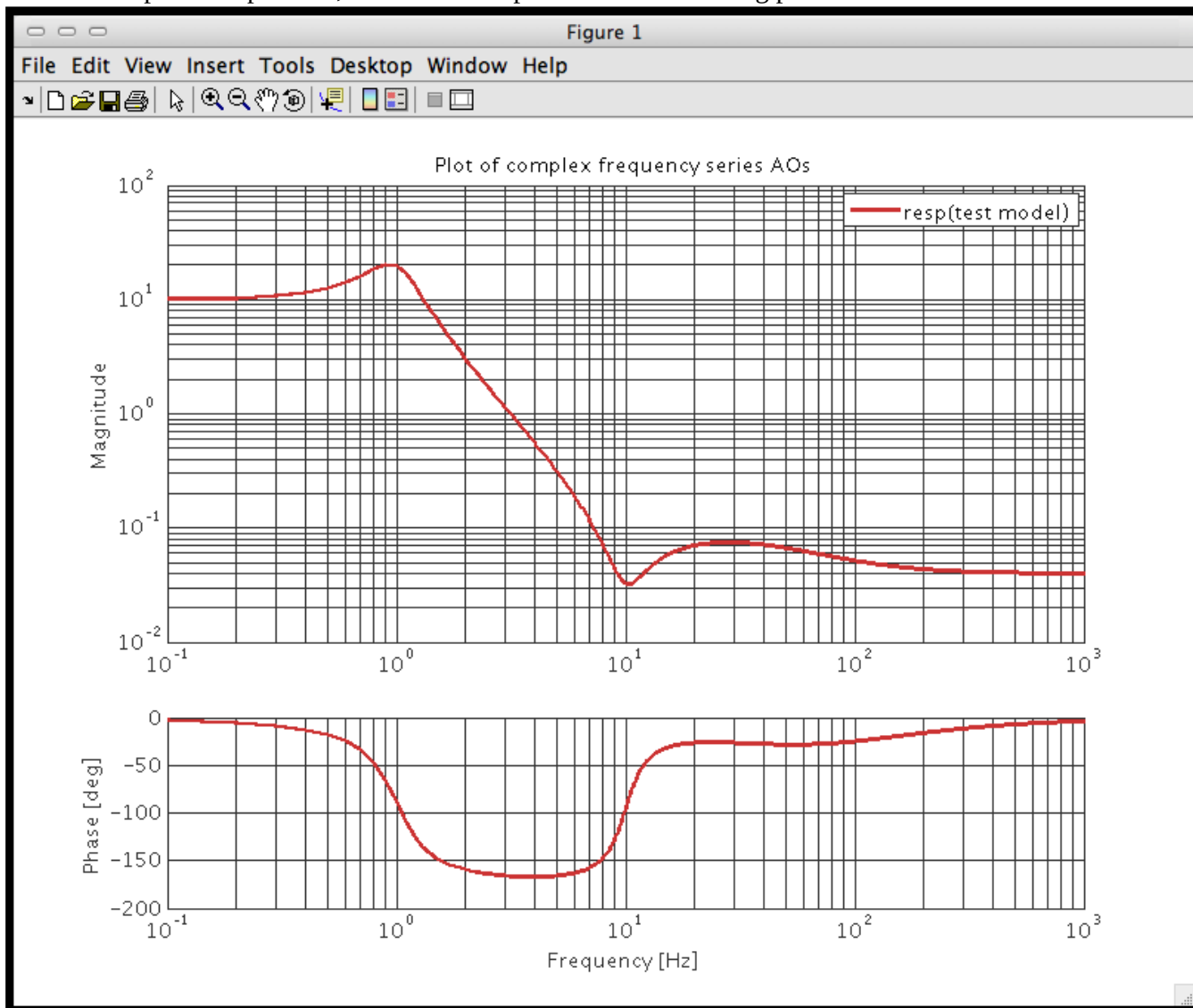
```
>> pzm = pzmodel(pl)
---- pzmodel 1 ----
model: test model
gain: 10
pole 001: pole(1,2)
pole 002: pole(40)
zero 001: zero(10,3)
zero 002: zero(100)
-----
```

Computing the response of the model

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

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

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



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

```
>> rpl = plist();
>> 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 1.5 2007/11/14
11:20:09 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

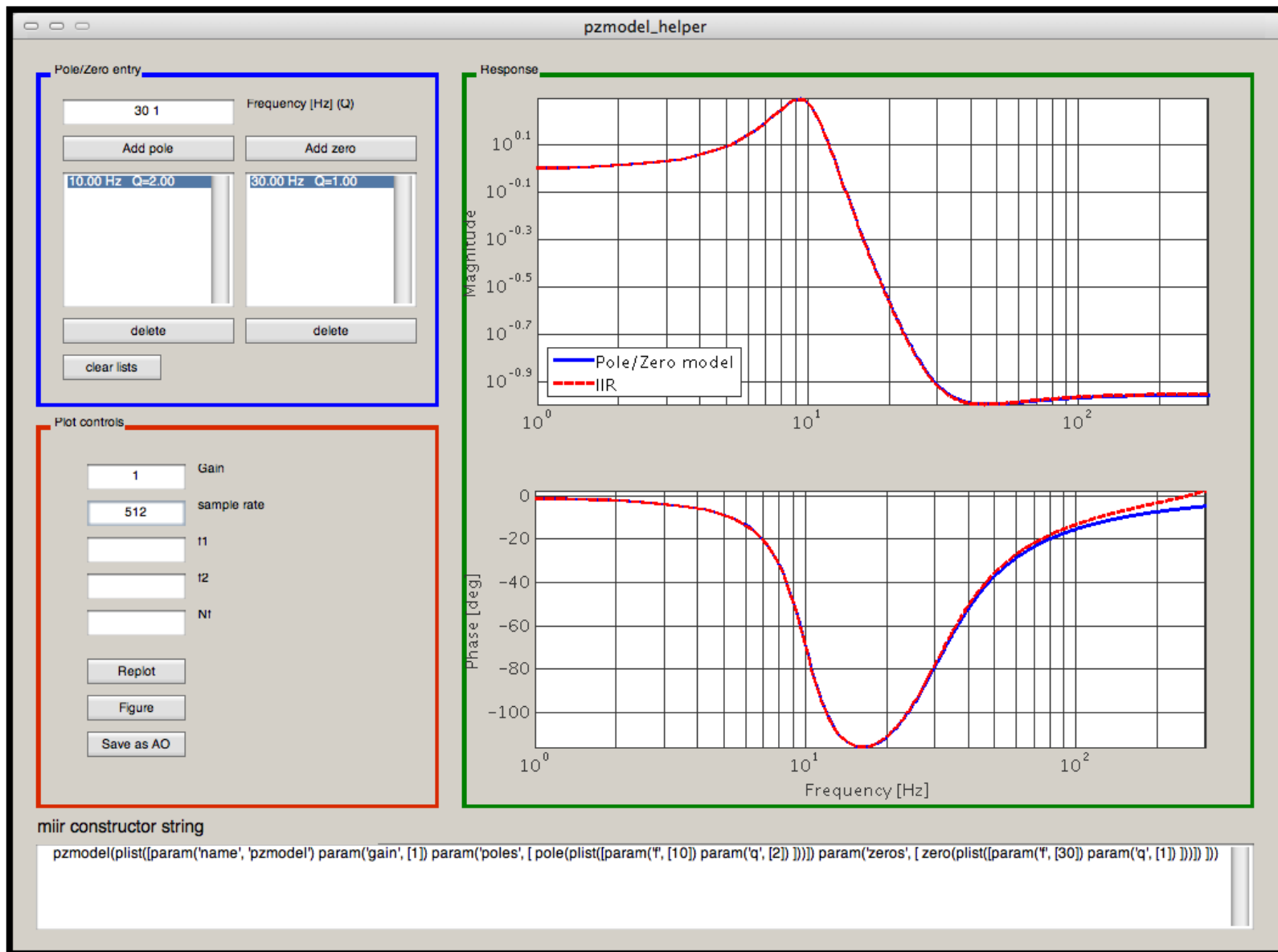
Model helper GUI 

©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

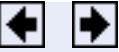
Pole/zero models can be converted to IIR filters using the bilinear transform. The result of the conversion is an `miir` object. To convert a model, you call the constructor of the `miir` class with a `plist` input:

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

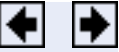
If no sample rate is specified, then the conversion is done for a sample rate equal to 8 times the highest pole or zero frequency. You can set the sample rate by specifying it in the parameter list:

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

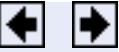
For more information of IIR filters in LTPDA, see [IIR Filters](#).



Standard Math Operations



Operators



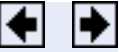
Signal Processing in LTPDA

Content needs written...

 Operators

Digital Filtering 

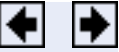
©LTP Team



Digital Filtering

Content needs written...

©LTP Team



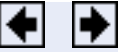
IIR Filters

Content needs written...

 Digital Filtering

FIR Filters 

©LTP Team



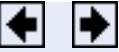
FIR Filters

Content needs written...

 IIR Filters

Spectral Estimation 

©LTP Team



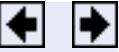
Spectral Estimation

Content needs written...

 [FIR Filters](#)

[Fitting Algorithms](#) 

©LTP Team



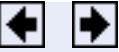
Fitting Algorithms

Content needs written...

 Spectral Estimation

Graphical User Interfaces in LTPDA 

©LTP Team



Graphical User Interfaces in LTPDA



gLTPDA

Content needs written...

 Graphical User Interfaces in LTPDA

LTPDA Object Browser 

©LTP Team

LTPDA Toolbox

[contents](#)



LTPDA Object Browser

Content needs written...

 [gLTPDA](#)

[LTPDA Repository Browser](#) 

©LTP Team

LTPDA Toolbox

[contents](#)



LTPDA Repository Browser

Content needs written...



LTPDA Object Browser

Working with an LTPDA Repository

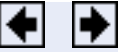


©LTP Team

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.



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`).

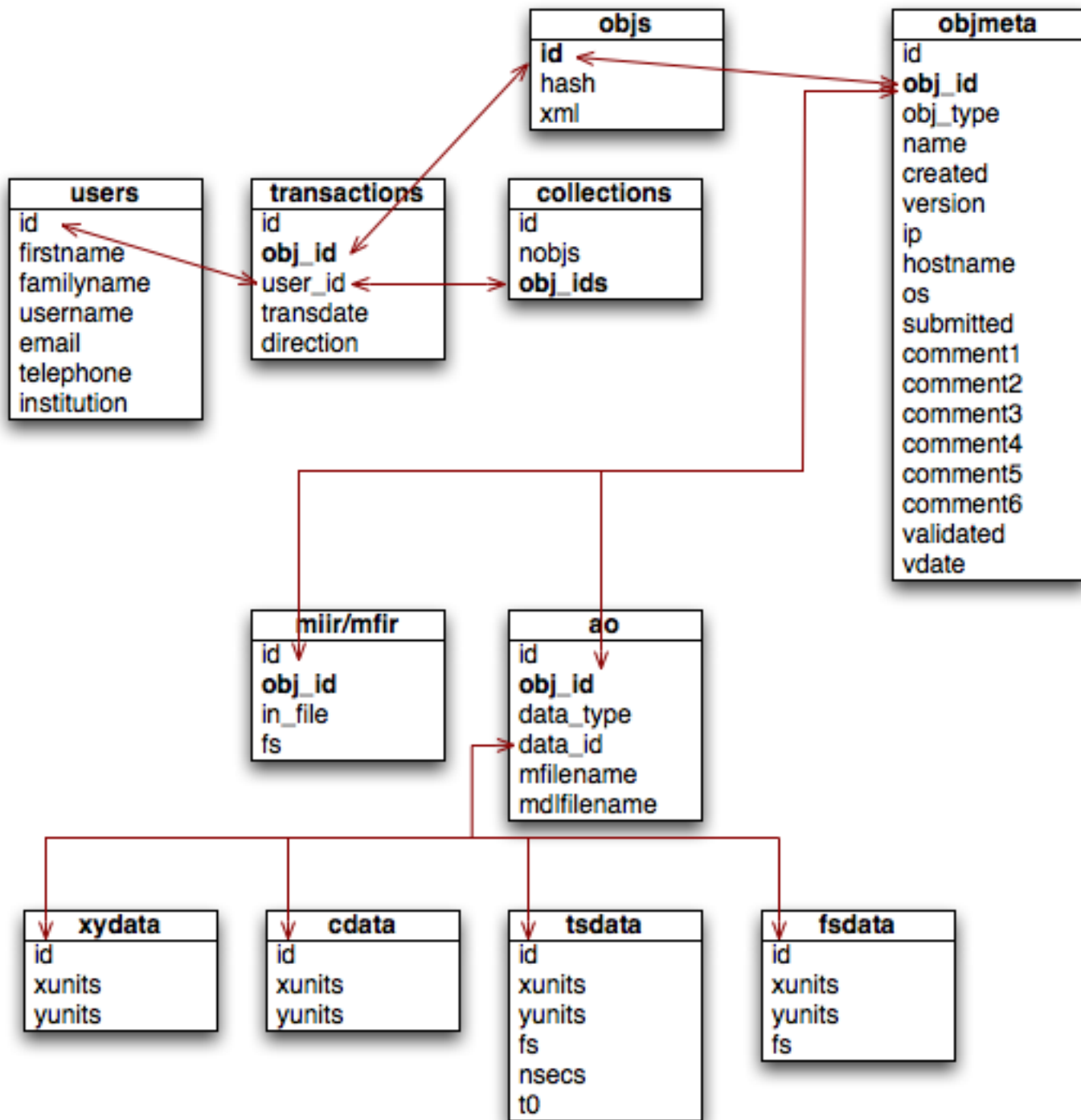
Any standard MySQL client can be used to query and search an LTPDA repository. For example, using a web-client or the standard MySQL command-line interface. In addition, the LTPDA Toolbox provides two ways to search the database: using the command `ltpda_dbquery` or using the LTPDA repository GUI. It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



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:

<code>object</code>	The LTPDA object to submit
<code>sinfo</code>	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);

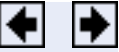
% Close connection
close(conn);
```

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 objects, as well as querying of a repository.

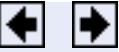
Connecting to a repository

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



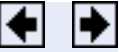
Setting up an LTPDA Repository -- administrator's guide

Content needs written...

 Using the LTPDA Repository GUI

Class descriptions 

©LTP Team



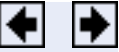
Class descriptions

Content needs written...

 Setting up an LTPDA Repository -- administrator's guide

ao Class 

©LTP Team



ao Class

Class description for ao class:

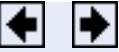
Properties	Description
tag	Unique tag for this AO
name	Name of AO
data	Data object associated with this AO
hist	History object associated with this AO
provenance	Provenance object associated with this AO
comment	A human-readable comment describing this AO
mfile	full text representation of the m-file 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

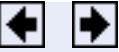


provenance Class



LTPDA Toolbox

[contents](#)



tsdata Class

 [provenance Class](#)

[fsdata Class](#) 

©LTP Team

LTPDA Toolbox

[contents](#)



fsdata Class

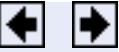


tsdata Class

xydata Class



©LTP Team



xydata Class

Content needs written...

 [fsdata Class](#)

[cdata Class](#) 

©LTP Team



cdata Class

Content needs written...

 [xydata Class](#)

[plist Class](#) 

©LTP Team



plist Class

Content needs written...

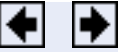


[cdata Class](#)

[param Class](#)



©LTP Team



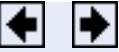
param Class

Content needs written...

 [plist Class](#)

[specwin Class](#) 

©LTP Team



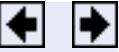
specwin Class

Content needs written...

 param Class

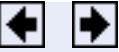
miir Class 

©LTP Team



miir Class

Content needs written...



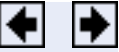
mfir Class

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...

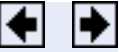


pzmodel Class

zero Class



©LTP Team

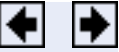


zero Class

Content needs written...

LTPDA Toolbox

[contents](#)



Functions - By Catagory



zero Class

Functions - Alphabetical List



©LTP Team

LTPDA Toolbox



Functions — Alphabetical List



Functions — by Catagory

Getting Started



©LTP Team

[Home](#) > [classes](#) > [@ao](#) > ao.m

ao

PURPOSE

AO analysis object class constructor.

SYNOPSIS

function varargout = ao(varargin)

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 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.dat')	- 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:														
a = ao('file',pl)	<table border="0"> <tr> <td>'type'</td> <td>'tsdata','fsdata','xydata'</td> </tr> <tr> <td></td> <td>default = 'tsdata'</td> </tr> <tr> <td>'use_fs'</td> <td>if this value is set, the x-axes is computed by the fs value.</td> </tr> <tr> <td>'columns'</td> <td>[1 2 1 4]</td> </tr> <tr> <td></td> <td>default = [1 2]</td> </tr> <tr> <td></td> <td>Each pair represented the x- and y-axes.</td> </tr> <tr> <td></td> <td>(Each column pair creates an analysis</td> </tr> </table>	'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
'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)

object

Is the value 'use_fs' is used then represent each column the y-axes. (Each column creates an analysis object)

'num_columns' Maximum columns in the file
default = 10

'comment_char' The comment character in the file
default = '%'

'description' To set the description in the analysis

'...' 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 A0 from a filename.
Example: `plist(param('filename', 'a1.xml'))`
- 'fcn' - construct an A0 from some valid MATLAB function.
Example: `plist(param('fcn', 'randn(100)'))`
- 'vals' - construct an A0 from a set of values.
Example: `plist(param('vals', [1 2 3]))`
- 'tsfcn' - construct an A0 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))'))])`
- 'win' - construct an A0 from a specwin window function. Tjis creates a cdata type A0 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 gausspuls)
'Square wave'    - 'f', 'duty' (help square)
'Sawtooth'       - 'f', 'width' (help sawtooth)
```

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')
```

See also `tsdata`, `fsdata`, `xydata`, `cdata`, `xyzdata`

M Hewitson 30-01-07

%%%

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.
- [hist](#) HIST overloads the histogram function (`hist`) of MATLAB for Analysis Objects.
- [isfield](#) ISFIELD tests if the given field is one of the object properties.
- [set](#) SET set an analysis object property.
- [xmlparse](#) XMLPARSE creates a ao from a ao xml file.

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.
- **aosplit** SPLIT an analysis object into the specified segments.
- **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.
- **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.
- **decimate** DECIMATE AOs containing time-series data.
- **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.
- **eig** EIG overloads the determinant function for Analysis objects.
- **exp** EXP overloads the exp operator for Analysis objects. Exponential.
- **fft** FFT overloads the fft operator for Analysis objects.
- **filter** FILTER overrides the filter function for analysis objects.
- **filtfilt** FILTFILT overrides the filtfilt function for analysis objects.
- **find** FIND particular samples that satisfy the input query and return a new AO.
- **get** GET get ao properties.
- **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.
- **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.
- **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.

SUBFUNCTIONS

- **function a = aoFromVals(vals, VERSION, ALGONAME)**
- **function a = aoFromWaveform(pl, VERSION, ALGONAME)**
- **function a = aoFromTSfcn(pl, VERSION, ALGONAME)**
- **function a = aoFromFcn(pl, VERSION, ALGONAME)**
- **function a = aoFromFilenameAndPlist(file_name, pl, VERSION, ALGONAME)**
- **function a = aoFromSpecWin(win, VERSION, ALGONAME)**
- **function bs = aoFromServer(args, VERSION, ALGONAME);**
- **function out = getDefaultPlist(varargin)**

SOURCE CODE

```

0001 function varargout = ao(varargin)
0002 % AO analysis object class constructor.
0003 %
0004 %     Create an analysis object.
0005 %
0006 %     Properties:
0007 %         name        - name of analysis object
0008 %         data         - data object (tsdata, fsdata)
0009 %         history      - history object
0010 %         provenance   - creation info, machine, user, etc
0011 %
0012 %
0013 %     Possible constructors:
0014 %
0015 %         a = ao()           - creates an empty analysis object
0016 %         a = ao('a1.xml')   - creates a new analysis object by loading the
0017 %                             analysis object from disk.
0018 %         a = ao('file.txt') - creates a new analysis object by loading the
0019 %         a = ao('file.dat')  data in 'file.txt'. The ascii file is assumed
0020 %                             to be an equally sampled two-column file of
0021 %                             time and amplitude. By default, the amplitude
0022 %                             units are taken to be Volts ('V') and the
0023 %                             time samples are assumed to be in seconds.
0024 %         a = ao('file',pl)  - creates a new analysis object by loading the

```

```

0025 % data in 'file'. The parameter list decide how the
0026 % analysis object is created. The valid key values
0027 % of the parameter list are:
0028 % 'type'          'tsdata','fsdata','xydata'
0029 %                  default = 'tsdata'
0030 % 'use_fs'        if this value is set, the
0031 %                  x-axes is computed by the fs value.
0032 % 'columns'       [1 2 1 4]
0033 %                  default = [1 2]
0034 %                  Each pair represented the x- and y-
axes.
0035 %                  (Each column pair creates an
analysis object)
0036 %                  Is the value 'use_fs' is used then
0037 %                  represent each column the y-axes.
0038 %                  (Each column creates an analysis
object)
0039 % 'num_columns'   Maximum columns in the file
0040 %                  default = 10
0041 % 'comment_char'  The comment character in the file
0042 %                  default = '%'
0043 % 'description'   To set the description in the
analysis object
0044 % '...'           every property of the data object
0045 %                  e.g. 'name'
0046 %                  If the constructor creates multiple ao's it is
0047 %                  possible to give each data class its own e.g.
0048 %                  'name'. In this case the parameter list with the
0049 %                  key 'name' must have cell of the different values
0050 %                  as the name of the different data objects. e.g.
0051 %                  pl = plist('columns', [1 2 1 3], ...
0052 %                             'name',    {'name1' 'name2'}, ...
0053 %                             'xunits',  'sec', ...
0054 %                             'yunits',  {'V' 'Hz'});
0055 %                  This parameter list creates two ao's with tsdata.
0056 %      a = ao(plist) - creates an analysis object with by reading
0057 %                  the file specified by the 'filename'
0058 %                  parameter in the plist.
0059 %      a = ao(data)  - creates an analysis object with a data
0060 %                  object. Data object can be one of tsdata,
0061 %                  fsdata, cdata, xydata, xyzdata.
0062 %      a = ao(data, hist) - creates an analysis object with a data
0063 %                  object and a history object
0064 %
0065 %      a = ao(specwin) - creates an analysis object from a specwin
0066 %                  object
0067 %
0068 % Parameters for plist constructor:
0069 %
0070 % 'filename' - construct an AO from a filename.
0071 %             Example: plist(param('filename', 'a1.xml'))
0072 %
0073 % 'fcn'      - construct an AO from some valid MATLAB function.

```



```

0074 %           Example: plist(param('fcn', 'randn(100)'))
0075 %
0076 % 'vals'      - construct an A0 from a set of values.
0077 %           Example: plist(param('vals', [1 2 3]))
0078 %
0079 % 'tsfcn'     - construct an A0 from a function of time. You need to also
0080 %               specify parameters 'fs' and 'nsecs' for this call.
0081 %           Example:
0082 %               plist([param('fs', 10) param('nsecs', 10) param('tsfcn',
0083 %               'sin(2*pi*1.4*t) + 0.1*randn(size(t))')])
0084 %
0085 % 'win'       - construct an A0 from a specwin window function. Tjis
0086 %               creates a cdata type A0 containing the window values.
0087 %           Example: plist(param('win', specwin('Hannning', 100)))
0088 %
0089 % 'waveform' - construct an A0 from a waveform description. Here you need
0090 %               to specify additional parameters 'fs' and 'nsecs', and also,
0091 %               for the following waveform types:
0092 %               'sine wave'      - 'f', 'phi'
0093 %               'noise'         - 'type' (can be 'Normal' or 'Uniform')
0094 %               'chirp'         - 'f0', 'f1', 't1' (help chirp)
0095 %               'Gaussian pulse' - 'f0', 'bw' (help gausspuls)
0096 %               'Square wave'   - 'f', 'duty' (help square)
0097 %               'Sawtooth'      - 'f', 'width' (help sawtooth)
0098 %
0099 % Examples:
0100 %
0101 % 1) Normally distributed random noise time-series
0102 %
0103 %     p      = plist('waveform', 'noise', 'fs', 10, 'nsecs', 1000);
0104 %     rd10   = ao(p);
0105 %
0106 %
0107 %
0108 % Indexing:
0109 %     b = a(1)           % where a is an array of analysis objects
0110 %     d = a.data;        % get the data object
0111 %     h = a.hist;        % get the history object
0112 %     d = a.data.x(1:20); % get a matrix of data values x;
0113 %
0114 % The following call returns a parameter list object that contains the
0115 % default parameter values:
0116 %
0117 % >> pl = ao(ao, 'Params')
0118 %
0119 %
0120 % See also tsdata, fsdata, xydata, cdata, xyzdata
0121 %
0122 % M Hewitson 30-01-07
0123 %
0124 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0125 %

```

```

0126 %         a = ao('object.mat') - creates a new analysis object by loading the
0127 %                               analysis object from disk .
0128
0129 ALGONAME = mfilename;
0130 VERSION  = '$Id: ao.m,v 1.56 2007/12/24 19:48:09 hewitson Exp $';
0131
0132 % Is this a 'Params' call?
0133 if (nargin == 2 || nargin == 3) &&...
0134     isa(varargin{1}, 'ao') &&...
0135     ischar(varargin{2})
0136     if strcmp(varargin{2}, 'Params')
0137         if nargin == 2
0138             varargout{1} = getDefaultPlist();
0139         else
0140             varargout{1} = getDefaultPlist(varargin{3});
0141         end
0142         return
0143     elseif strcmp(varargin{2}, 'Version')
0144         varargout{1} = VERSION;
0145         return
0146     end
0147 end
0148
0149
0150 % process input arguments
0151 args = [];
0152 k = 0;
0153 pls = [];
0154 for j=1:nargin
0155     if ~isempty(varargin{j})
0156         k = k + 1;
0157         if isa(varargin{j}, 'plist')
0158             pls = [pls varargin{j}];
0159         else
0160             args(k).val = varargin{j};
0161             args(k).n    = j;
0162         end
0163     end
0164 end
0165
0166 % We could have multiple input plist objects. Combine them here.
0167 if isa(pls, 'plist')
0168     pl = combine(pls);
0169 else
0170     pl = [];
0171 end
0172
0173 Nargs = length(args);
0174 if ~isempty(pl)
0175     args(Nargs+1).val = pl;
0176     args(Nargs+1).n    = j+1;
0177     Nargs = length(args);
0178 end

```

```

0179
0180 %-----
0181 % create from SERVER
0182 if Nargs > 0
0183     if ischar(args(1).val)
0184         if strcmp(args(1).val, 'SERVER')
0185             a = aoFromServer(args, VERSION, ALGONAME);
0186             varargout{1} = a;
0187             return
0188         end
0189     end
0190 end
0191
0192 %-----
0193 % create empty AO
0194 if Nargs == 0
0195     a.name          = 'None';
0196     a.data          = {};
0197     a.hist          = history(ALGONAME, VERSION);
0198     a.provenance    = provenance;
0199     a.description   = '';
0200     a.mfile         = {};
0201     a.mfilename     = '';
0202     a.mdlfile       = '';
0203     a.mdlfilename   = '';
0204     a.version       = VERSION;
0205     a.created       = time;
0206     varargout{1} = class(a, 'ao');
0207 %-----
0208 % copy AO, load from file, create from x-data
0209 elseif Nargs == 1 % create ao with data
0210     %%%%%%%%% Create from XML fragment %%%%%%%%%
0211     if isa(args(1).val, 'org.apache.xerces.dom.DeferredElementImpl')
0212         varargout{1} = fromxml(args(1).val);
0213     %%%%%%%%% From File %%%%%%%%%
0214     elseif ischar(args(1).val)
0215
0216         filename = args(1).val;
0217         [path, name, ext, vers] = fileparts(filename);
0218         switch ext
0219             case '.mat'
0220                 a = load(filename);
0221                 a = a.a;
0222             case '.xml'
0223                 a = xmlparse(ao, filename);
0224             case {' .txt', '.dat'}
0225                 a = aoFromFilenameAndPlist(args(1).val, plist, VERSION, ALGONAME);
0226             otherwise
0227                 error('### Unknown file type. ');
0228         end
0229         varargout{1} = a;
0230

```

```

0231 %-----
0232 % copy AO
0233 elseif isa(args(1).val, 'ao')
0234     varargout{1} = args(1).val;
0235 %-----
0236 % create from struct
0237 elseif isstruct(args(1).val)
0238     astruct = args(1).val;
0239     a.name = astruct.name;
0240     data = astruct.data;
0241     if isstruct(data)
0242         if isfield(data, {'t', 'x'})
0243             data = tsdata(data);
0244         elseif isfield(data, {'f', 'xx'})
0245             data = fsdata(data);
0246         elseif isfield(data, {'vals'})
0247             data = cdata(data);
0248         elseif isfield(data, {'x', 'y'})
0249             data = xydata(data);
0250         elseif isfield(data, {'x', 'y', 'z'})
0251             data = xyzdata(data);
0252         else
0253             error('### Unknown data type in input structure.');
```

```

0254         end
0255     end
0256     a.data = data;
0257     hist = astruct.hist;
0258     if isstruct(hist)
0259         hist = history(hist);
0260     end
0261     a.hist = hist;
0262     prov = astruct.provenance;
0263     if isstruct(prov)
0264         prov = provenance(prov);
0265     end
0266     a.provenance = prov;
0267     a.description = astruct.description;
0268     a.mfile = astruct.mfile;
0269     a.mfilename = astruct.mfilename;
0270     a.mdlfile = astruct.mdlfile;
0271     a.mdlfilename = astruct.mdlfilename;
0272     a.version = astruct.version;
0273     created = astruct.created;
0274     if isstruct(created)
0275         created = time(created);
0276     end
0277     a.created = created;
0278     varargout{1} = class(a, 'ao');
```

```

0279 %-----
0280 % create from constant
0281 elseif isnumeric(args(1).val)
```

```

0283
0284     a.name = 'const';
0285     a.data = cdata(args(1).val);
0286     a.hist = history(ALGONAME, VERSION, plist(param('vals', args(1).val)));
0287 %     a.hist = history(ALGONAME, VERSION, plist(param('fcn', [' sprintf('%g
', args(1).val) ']))));
0288     a.provenance = provenance;
0289     a.description = '';
0290     a.mfile = {};
0291     a.mfilename = '';
0292     a.mdlfile = '';
0293     a.mdlfilename = '';
0294     a.version = VERSION;
0295     a.created = time;
0296     varargout{1} = class(a, 'ao');
0297
0298     %-----
0299     % create data using plist
0300 elseif isa(args(1).val, 'plist')
0301
0302     pl = args(1).val;
0303     filename = find(pl, 'filename');
0304     fcn = find(pl, 'fcn');
0305     vals = find(pl, 'vals');
0306     tsfcn = find(pl, 'tsfcn');
0307     win = find(pl, 'win');
0308     waveform = find(pl, 'waveform');
0309
0310     if ~isempty(filename)
0311         % do filename constructor
0312         [path, name, ext, vers] = fileparts(filename);
0313         switch ext
0314             case '.mat'
0315                 a = load(filename);
0316                 varargout{1} = a.a;
0317             case '.xml'
0318                 varargout{1} = xmlparse(ao, filename);
0319             case {' .txt', '.dat'}
0320                 varargout{1} = aoFromFilenameAndPlist(filename, pl, VERSION,
ALGONAME);
0321             otherwise
0322                 error('### Unknown file type. ');
0323         end
0324
0325     elseif ~isempty(fcn)
0326         % do function constructor
0327         varargout{1} = aoFromFcn(pl, VERSION, ALGONAME);
0328
0329     elseif ~isempty(vals)
0330         % do vals constructor
0331         varargout{1} = aoFromVals(vals, VERSION, ALGONAME);
0332
0333

```

```

0333     elseif ~isempty(tsfcn)
0334         % do tsfcn constructor
0335         varargout{1} = aoFromTSfcn(pl, VERSION, ALGONAME);
0336
0337     elseif ~isempty(win)
0338         % do win constructor
0339         varargout{1} = aoFromSpecWin(win, VERSION, ALGONAME);
0340
0341     elseif ~isempty(waveform)
0342         % do waveform constructor
0343         varargout{1} = aoFromWaveform(pl, VERSION, ALGONAME);
0344
0345     else
0346         error('### Unknown AO constructor method.');
```

```

0349 % create from spectral window
0350 elseif isa(args(1).val, 'specwin')
0351     varargout{1} = aoFromSpecWin(args(1).val, VERSION, ALGONAME);
0352 %-----
0353 % create from data object
0354 else
0355     data          = args(1).val;
0356     a.name        = 'None';
0357     a.data        = data;
0358     a.hist        = history(ALGONAME, VERSION);
0359     a.provenance  = provenance;
0360     a.description = '';
0361     a.mfile       = {};
0362     a.mfilename   = '';
0363     a.mdlfile     = '';
0364     a.mdlfilename = '';
0365     a.version     = VERSION;
0366     a.created     = time;
0367     varargout{1} = class(a, 'ao');
0368     if ~isa(a.data, 'tsdata') &&...
0369         ~isa(a.data, 'fsdata') &&...
0370         ~isa(a.data, 'cdata') &&...
0371         ~isa(a.data, 'xydata') &&...
0372         ~isa(a.data, 'xyzdata')
0373         error('### unknown data object.')
0374     end
0375 end
0376 end
0377 %-----
0378 % data, hist
0379 % 'file_name.txt', pl
0380 elseif Nargs == 2
0381
0382     %%%%%%%%% Database
0383     if isa(varargin{1}, 'database')
0384         varargout{1} = retrieve(varargin{1}, varargin{2:end});
```

```

0385
0386 %%%%%%%%%%%%% 'file_name.txt', pl
0387 elseif ischar(args(1).val) && isa(args(2).val, 'plist')
0388
0389     ao_out = aoFromFilenameAndPlist(args(1).val, args(2).val, VERSION,
ALGONAME);
0390
0391     if nargout == 1
0392         varargout{1} = ao_out;
0393     elseif nargout == length(ao_out)
0394         for ii = 1:length(ao_out)
0395             varargout{ii} = ao_out(ii);
0396         end
0397     else
0398         warning ('### Too few output variables --> using only the first. ');
0399         varargout{1} = ao_out;
0400     end
0401
0402 % data, hist
0403 else
0404
0405     a.name          = 'None';
0406     a.data          = args(1).val;
0407     a.hist          = args(2).val; %history(ALGONAME, VERSION, [], varargin{2});
0408     a.provenance    = provenance;
0409     a.description    = '';
0410     a.mfile         = {};
0411     a.mfilename     = '';
0412     a.mdlfile       = '';
0413     a.mdlfilename   = '';
0414     a.version       = VERSION;
0415     a.created       = time;
0416     varargout{1}    = class(a, 'ao');
0417     if ~isa(a.data, 'tsdata') &&...
0418         ~isa(a.data, 'fsdata') &&...
0419         ~isa(a.data, 'cdata') &&...
0420         ~isa(a.data, 'xydata')
0421         error('### unknown data object.')
0422     end
0423     if ~isa(a.hist, 'history')
0424         error('### unknown history object.')
0425     end
0426
0427 end
0428
0429
0430 %-----
0431 % full constructor - all fields
0432 elseif Nargs == 9
0433
0434     a.name          = args(1).val;
0435     a.data          = args(2).val;
0436     a.hist          = args(3).val;

```

```

0437     a.provenance = args(4).val;
0438     a.description = args(5).val;
0439     a.mfile       = args(6).val;
0440     a.mfilename   = args(7).val;
0441     a.mdlfile     = args(8).val;
0442     a.mdlfilename = args(9).val;
0443     a.version     = VERSION;
0444     a.created      = time;
0445     varargout{1} = class(a, 'ao');
0446
0447 else
0448     error('### Unknown number of constructor arguments.');
```

```

0449 end
0450
0451 %-----
0452 % construct an ao from a value set
0453 %
0454 function a = aoFromVals(vals, VERSION, ALGONAME)
0455
0456 % Make a cdata object
0457 ts = cdata(plist(param('vals', vals)));
0458 ts = set(ts, 'name', 'Data');
```

```

0459
0460 % Make an analysis object
0461 a.name = 'Data';
0462 a.data = ts;
0463 a.hist = history(ALGONAME, VERSION, plist(param('vals', vals)));
0464 a.provenance = provenance;
0465 a.description = '';
0466 a.mfile       = {};
0467 a.mfilename   = '';
0468 a.mdlfile     = '';
0469 a.mdlfilename = '';
0470 a.version     = VERSION;
0471 a.created      = time;
0472 a = class(a, 'ao');
```

```

0473
0474
0475
0476 %-----
0477 % construct an ao from a waveform
0478 %
0479 function a = aoFromWaveform(pl, VERSION, ALGONAME)
0480
0481 waveform = find(pl, 'waveform');
```

```

0482 switch waveform
0483     %----- Sine Wave
0484     case 'sine wave'
0485         freq = find(pl, 'f');
0486         phi  = find(pl, 'phi');
0487         tsfcn = sprintf('sin(2*pi*%g*t + %g*pi/180)', freq, phi);
0488     %----- Noise
```



```

0489 case 'noise'
0490     ntype = find(pl, 'type');
0491     if isempty(ntype)
0492         ntype = 'Normal';
0493     end
0494     switch ntype
0495         case 'Normal'
0496             tsfcn = 'randn(size(t))';
0497         case 'Uniform'
0498             tsfcn = 'rand(size(t))';
0499     end
0500     %----- Chirp
0501 case 'chirp'
0502     f0 = find(pl, 'f0');
0503     fe = find(pl, 'f1');
0504     te = find(pl, 't1');
0505     tsfcn = sprintf('chirp(t,%g,%g,%g)', f0, fe, te);
0506     %----- Gaussian pulse
0507 case 'Gaussian pulse'
0508     fc = find(pl, 'f0');
0509     bw = find(pl, 'bw');
0510     tsfcn = sprintf('gauspuls(t,%g,%g)', fc, bw);
0511 case 'Square wave'
0512     freq = find(pl, 'f');
0513     duty = find(pl, 'duty');
0514     tsfcn = sprintf('square(2*pi*%g*t,%g)', freq, duty);
0515 case 'Sawtooth'
0516     freq = find(pl, 'f');
0517     width = find(pl, 'width');
0518     tsfcn = sprintf('sawtooth(2*pi*%g*t,%g)', freq, width);
0519 end
0520
0521 nsecs = find(pl, 'nsecs');
0522 fs     = find(pl, 'fs');
0523
0524 % construct tsdata
0525 t = linspace(0, nsecs-1/fs, nsecs*fs);
0526 % make y data
0527 eval(['y=' tsfcn ';']);
0528
0529 ts = tsdata(t,y);
0530 ts = set(ts, 'name', tsfcn);
0531 ts = set(ts, 'xunits', 's');
0532 ts = set(ts, 'yunits', 'V');
0533
0534 % Make an analysis object
0535 a.name = waveform;
0536 a.data = ts;
0537 a.hist = history(ALGONAME, VERSION, pl);
0538 a.provenance = provenance;
0539 a.description = '';

```

```

0540 a.mfile      = {};
0541 a.mfilename   = '';
0542 a.mdlfile     = '';
0543 a.mdlfilename = '';
0544 a.version     = VERSION;
0545 a.created     = time;
0546 a = class(a, 'ao');
0547
0548
0549
0550 %-----
0551 % construct an ao from a ts-function string
0552 %
0553 function a = aoFromTSfcn(pl, VERSION, ALGONAME)
0554
0555 nsecs          = find(pl, 'nsecs');
0556 fs             = find(pl, 'fs');
0557 fcn            = find(pl, 'tsfcn');
0558 randn_state    = find(pl, 'rand_state');
0559
0560 % construct tsdata
0561 t = linspace(0, nsecs-1/fs, nsecs*fs);
0562
0563 % make y data
0564
0565 % If the randn state is in the parameter list then set the randn function to
0566 % this state
0567 if ~isempty(randn_state)
0568     randn('state', randn_state);
0569 else % Store the state of the random function in the parameter list
0570     randn_state = randn('state');
0571     pl = append(pl, 'rand_state', randn_state);
0572 end
0573
0574 y = eval([fcn ';']);
0575
0576 ts = tsdata(t,y);
0577 ts = set(ts, 'name', fcn);
0578 ts = set(ts, 'xunits', 's');
0579 ts = set(ts, 'yunits', 'V');
0580
0581 % Make an analysis object
0582 a.name = 'TSfcn';
0583 a.data = ts;
0584 a.hist = history(ALGONAME, VERSION, pl);
0585 a.provenance = provenance;
0586 a.description = '';
0587 a.mfile      = {};
0588 a.mfilename   = '';
0589 a.mdlfile     = '';
0590 a.mdlfilename = '';
0591 a.version     = VERSION;

```

```

0592 a.created      = time;
0593 a = class(a, 'ao');
0594
0595 %-----
0596 % construct an ao from a function string
0597 %
0598 function a = aoFromFcn(pl, VERSION, ALGONAME)
0599
0600 fcn = find(pl, 'fcn');
0601
0602 % Make a cdata object
0603 ts = cdata(plist(param('fcn', fcn)));
0604 ts = set(ts, 'name', fcn);
0605
0606 % Make an analysis object
0607 a.name = fcn;
0608 a.data = ts;
0609 a.hist = history(ALGONAME, VERSION, pl);
0610 a.provenance = provenance;
0611 a.description = '';
0612 a.mfile       = {};
0613 a.mfilename   = '';
0614 a.mdlfile     = '';
0615 a.mdlfilename = '';
0616 a.version     = VERSION;
0617 a.created     = time;
0618 a = class(a, 'ao');
0619
0620 %-----
0621 % construct an ao from filename AND parameter list
0622 %
0623 function a = aoFromFilenameAndPlist(file_name, pl, VERSION, ALGONAME)
0624
0625     a = [];
0626
0627 %     disp([' + creating AO(s) from text file: ' file_name]);
0628
0629     [pathstr, f_name, ext] = fileparts(file_name);
0630
0631     %%%%%%%%%% Define default parameter list %%%%%%%%%%
0632     pl_default = plist('columns',    [1 2],    ...
0633                        'num_columns', 10,      ...
0634                        'type',        'tsdata', ...
0635                        'use_fs',      '',       ...
0636                        'comment_char', '%',     ...
0637                        'xunits',      'sec',    ...
0638                        'yunits',      'V');
0639     pl = combine(pl, pl_default);
0640
0641     pl = pset(pl, param('filename', [f_name ext]));
0642     pl = pset(pl, param('filepath', pathstr));
0643

```

```

0644 data_type      = find (pl, 'type');
0645 columns        = find (pl, 'columns');
0646 num_columns     = find (pl, 'num_columns');
0647 comment_char    = find (pl, 'comment_char');
0648 use_fs          = find (pl, 'use_fs');
0649
0650 %%%%%%%%%%%%% read file %%%%%%%%%%%%%%
0651 [fid,msg] = fopen (file_name, 'r');
0652 if (fid < 0)
0653     error ('### can not open file: %s \n\n### error msg:',file_name, msg);
0654 end
0655
0656 %%%%%%%%%%%%% create scan format: '%f %f %f %f %f' %%%%%%%%%%%%%%
0657 scan_format = [];
0658 for ii=1:num_columns
0659     scan_format = [scan_format '%f '];
0660 end
0661 scan_format = scan_format(1:end-1); % remove the last whitespace
0662
0663 %%%%%%%%%%%%% scan the file %%%%%%%%%%%%%%
0664 raw_data = textscan( fid, scan_format, ...
0665                     'commentStyle', comment_char, ...
0666                     'CollectOutput', 1);
0667 f_data = raw_data{1};
0668
0669 if isempty(f_data)
0670     error([char(10) '### There are no data. ' ...
0671           'Did you use the right comment character?' char
0672           '### The current comment character is: ' comment_char char
0673           '### Use a parameter list with the parameter:' char
0674           '### plist('comment_char', '%')']);
0675 end
0676
0677 fclose(fid);
0678
0679 %%%%%%%%%%%%% Create for each column pair the data object %%%%%%%%%%%%%%
0680 if isempty(use_fs)
0681
0682     %%%%%%%%%%%%% The numbers in columns must be straight %%%%%%%%%%%%%%
0683     if mod(length(columns),2) ~= 0
0684         error('### the numbers in columns must be straight');
0685     end
0686
0687     for lauf = 1:length(columns)/2
0688
0689         data_x_axes = f_data(:, columns(lauf*2-1));
0690         data_y_axes = f_data(:, columns(lauf*2));
0691
0692         % create data object corresponding to the parameter list

```

```

0693     ao_data = [];
0694     switch data_type
0695     case 'tsdata'
0696         ao_data = tsdata( data_x_axes, data_y_axes);
0697     case 'fsdata'
0698         ao_data = fsdata( data_x_axes, data_y_axes);
0699     case 'cdata'
0700         error('### please code me up')
0701     case 'xydata'
0702         ao_data = xydata( data_x_axes, data_y_axes);
0703     otherwise
0704         error('### unknown data type '%s'', data_type);
0705     end
0706     ao_data.name = sprintf('f_name_%02d', lauf);
0707
0708     aa      = [];
0709     aa.name = sprintf('f_name_%02d', lauf);
0710     aa.data = ao_data;
0711     aa.hist = history(ALGONAME, VERSION, pl);
0712     aa.provenance = provenance;
0713
0714     %% Set the description from the parameter list or from the file
0715     description_pl = find(pl, 'description');
0716     if ~isempty(description_pl)
0717         aa.description = description_pl;
0718     else
0719         aa.description = '';
0720     end
0721
0722     aa.mfile      = {};
0723     aa.mfilename  = '';
0724     aa.mdlfile    = '';
0725     aa.mdlfilename = '';
0726     aa.version    = VERSION;
0727     aa.created    = time;
0728     aa = class(aa, 'ao');
0729
0730     a = [a aa];
0731
0732     end
0733
0734     %%%%%%%%% Create for each column AND fs a data object %%%%%%%%%
0735     else % isempty(use_fs)
0736
0737         for lauf = 1:length(columns)
0738
0739             data_y_axes = f_data(:, columns(lauf));
0740
0741             %% create data object corresponding to the parameter list
0742             ao_data = [];
0743             switch data_type
0744             case 'tsdata'
0745                 ao_data = tsdata(data_y_axes, use_fs);

```

```

0746         case 'fsdata'
0747             ao_data = fsdata(data_y_axes, use_fs);
0748         case 'cdata'
0749             error('### please code me up')
0750         case 'xydata'
0751             error('### please code me up')
0752         otherwise
0753             error('### unknown data type '%s'', data_type);
0754     end
0755     ao_data.name = sprintf('f_name_%02d', lauf);
0756
0757     aa      = [];
0758     aa.name = sprintf('f_name_%02d', lauf);
0759     aa.data = ao_data;
0760     aa.hist = history(ALGONAME, VERSION, pl);
0761     aa.provenance = provenance;
0762
0763     %% Set the description from the parameter list or from the file
0764     description_pl = find(pl, 'description');
0765     if ~isempty(description_pl)
0766         aa.description = description_pl;
0767     else
0768         aa.description = '';
0769     end
0770
0771     aa.mfile      = {};
0772     aa.mfilename  = '';
0773     aa.mdlfile    = '';
0774     aa.mdlfilename = '';
0775     aa.version    = VERSION;
0776     aa.created    = time;
0777     aa = class(aa, 'ao');
0778
0779     a = [a aa];
0780
0781 end
0782
0783 end
0784
0785 %%%%%%%%%%% set fields of the data object from the parameter list %%%%%%%%%%%
0786 %
0787 for ii = 1:length(a)
0788     fields = fieldnames(a(ii).data);
0789
0790     for jj = 1:length(pl.params)
0791         parameter = pl.params(jj);
0792
0793         if ismember(parameter.key, fields)
0794
0795             if iscell(parameter.val)
0796                 if length(parameter.val) > 1
0797                     if length(parameter.val) ~= length(a)
0798                         warning([sprintf('The values of the key '%s'' ', parameter.

```

```

key) ...
0798                                     'have not the same length as the lenthg of
' ...
0799                                     'the output. Using the first value!']);
0800         a(ii).data.(parameter.key) = parameter.val{1};
0801     else
0802         a(ii).data.(parameter.key) = parameter.val{ii};
0803     end
0804 else
0805     a(ii).data.(parameter.key) = parameter.val{1};
0806 end
0807 else
0808     a(ii).data.(parameter.key) = parameter.val;
0809 end
0810
0811     end % is parameter key member of the data object
0812 end % for-loop over all parameter
0813 end % for-loop over all ao's
0814
0815
0816 %-----
0817 % construct an ao from a Spectral window
0818 %
0819 function a = aoFromSpecWin(win, VERSION, ALGONAME)
0820
0821 % Make a cdata object
0822 ts = cdata(win.win);
0823 ts = set(ts, 'name', win.name);
0824
0825 % Make an analysis object
0826 a.name = win.name;
0827 a.data = ts;
0828 a.hist = history(ALGONAME, VERSION, plist(param('win', win)));
0829 a.provenance = provenance;
0830 a.description = '';
0831 a.mfile = {};
0832 a.mfilename = '';
0833 a.mdlfile = '';
0834 a.mdlfilename = '';
0835 a.version = VERSION;
0836 a.created = time;
0837 a = class(a, 'ao');
0838
0839
0840
0841 %-----
0842 % construct an ao from server
0843 %
0844 function bs = aoFromServer(args, VERSION, ALGONAME);
0845
0846 % retrieve ao from server
0847 bs = retrieve(args(2).val, [args(3:end).val]);

```

```

0848
0849
0850 %-----
0851 % Default Parameter Lists
0852 %
0853 function out = getDefaultPlist(varargin)
0854
0855 % list of available parameter sets
0856 sets = {'From XML file', 'From ASCII file', 'Function', 'Values',...
0857         'Time-series Function', 'From Window', 'Waveform'};
0858
0859 if nargin == 0
0860     out = sets;
0861     return
0862 end
0863
0864 set = varargin{1};
0865
0866 switch set
0867     case 'From XML file'
0868         out = plist('filename', 'foo.xml');
0869     case 'From ASCII file'
0870         out = plist('filename', 'foo.txt',...
0871                     'type', 'tsdata',...
0872                     'columns', [1 2],...
0873                     'num_columns', 10,...
0874                     'comment_char', '%');
0875     case 'Function'
0876         out = plist('fcn', 'randn(100,1)');
0877     case 'Values'
0878         out = plist('vals', [1 2 3]);
0879     case 'Time-series Function'
0880         out = plist('tsfcn', 'randn(size(t))', 'fs', 10, 'nsecs', 100);
0881     case 'From Window'
0882         out = plist('win', specwin('Hanning', 100));
0883     case 'Waveform'
0884         out = plist('waveform', 'sine wave',...
0885                     'f', 1.23,...
0886                     'phi', 0,...
0887                     'fs', 10,...
0888                     'nsecs', 10);
0889     otherwise
0890         out = plist();
0891 end
0892
0893
0894 % END

```


[Home](#) > [classes](#) > [@plist](#) > plist.m

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

VERSION: \$Id: plist.m,v 1.15 2007/12/12 18:45:09 ingo Exp \$

HISTORY: 30-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a parameter list into a string.
- [plist](#) PLIST Plist class object constructor.
- [xmlparse](#) XMLPARSE creates a parameter list from a plist xml file.

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.
- **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.
- **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.

SUBFUNCTIONS

- **function pl = init()**

SOURCE CODE

```

0001 function pl = plist(varargin)
0002 % PLIST Plist class object constructor.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: PLIST Plist class object constructor.
0007 %
0008 % CONSTRUCTORS:
0009 %
0010 %         pl = plist()                - create an empty plist object.
0011 %         pl = plist(p)              - create a plist with elements p
0012 %                                     where p is an array of param
objects.
0013 %         pl = plist('key1, val1, 'key2', 'val2') - create a plist with the
0014 %                                     key/values pairs
0015 %
0016 % VERSION:      $Id: plist.m,v 1.15 2007/12/12 18:45:09 ingo Exp $
0017 %
0018 % HISTORY:      30-01-07 M Hewitson
0019 %                Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

0022
0023 VERSION = '$Id: plist.m,v 1.15 2007/12/12 18:45:09 ingo Exp $';
0024
0025 %%%% Check if this is a call for parameters %%%%
0026 if nargin == 2
0027     if isa(varargin{1}, 'plist') && ischar(varargin{2})
0028         in = char(varargin{2});
0029         if strcmp(in, 'Params')
0030             pl = plist;
0031             return
0032         elseif strcmp(in, 'Version')
0033             pl = VERSION;
0034             return
0035         end
0036     end
0037 end
0038
0039 %%%%%%%%%% define parameter properties list %%%%%%%%%%
0040
0041 function pl = init()
0042     pl.name      = 'plist object';
0043     pl.params    = [];
0044     pl.version   = VERSION;
0045     pl.created   = time;
0046     pl = class(pl, 'plist');
0047 end
0048
0049 %%%%%%%%%% Create parameter list object %%%%%%%%%%
0050
0051 %%%%%%%%%% pl = plist() %%%%%%%%%%
0052 % create default parameter list object
0053 if nargin == 0
0054
0055     pl = init();
0056
0057 elseif nargin == 1
0058
0059     %%%%%%%%%% Create from XML fragment %%%%%%%%%%
0060     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0061         pl = fromxml(varargin{1});
0062
0063     %%%%%%%%%% p = param(plist) %%%%%%%%%%
0064     elseif isa(varargin{1}, 'plist')
0065         pl = varargin{1};
0066
0067     %%%%%%%%%% pl = plist(struct) %%%%%%%%%%
0068     elseif isstruct(varargin{1})
0069         plstruct = varargin{1};
0070
0071         pi = plstruct.params;
0072         params = [];
0073         for j=1:length(pi)

```

```

0074     if isstruct(pi(j))
0075         params = [params param(pi(j))];
0076     elseif isa(pi(j), 'param')
0077         params = [params pi(j)];
0078     else
0079         error ('### the parameter is not from class param.')
0080     end
0081 end
0082
0083 pl      = init();
0084 pl.name  = plstruct.name;
0085 pl.created = plstruct.created;
0086 if isstruct(pl.created)
0087     pl.created = time(pl.created);
0088 end
0089 pl.params = params;
0090 pl.version = plstruct.version;
0091
0092 %%%%%%%%%%% From file %%%%%%%%%%%
0093 elseif ischar(varargin{1})
0094
0095     filename = varargin{1};
0096     [path, name, ext, vers] = fileparts(filename);
0097     switch ext
0098     case '.mat'
0099         pl = load(filename);
0100         pl = pl.a;
0101     case '.xml'
0102         pl = xmlparse(plist, filename);
0103     otherwise
0104         error('### Unknown file type.');
```

```

0105     end
0106 %%%%%%%%%%% pl = plist(param) %%%%%%%%%%%
0107 elseif isa(varargin{1}, 'param')
0108     pl      = init();
0109
0110     %% Use only upper case characters for the 'key'
0111     param_list = varargin{1};
0112     for kk = 1:length(param_list)
0113         param_list(kk).key = upper(param_list(kk).key);
0114     end
0115
0116     pl.params = param_list;
0117
0118 else
0119     error ('### unknown arguments to construct a parameter list')
0120 end
0121 elseif nargin == 2
0122 %%%%%%%%%%% From DATABASE %%%%%%%%%%%
0123 if isa(varargin{1}, 'database')
0124
0125     pl = retrieve(varargin{1}, varargin{2:end});

```

```

0126
0127     %%%%%%%%%%    pl = plist('key1', val1)    %%%%%%%%%%
0128 else
0129     pl = init();
0130     key = upper(varargin{1});
0131     val =         varargin{2};
0132
0133     pl.params = param(key, val);
0134 end
0135
0136 elseif nargin > 2
0137
0138     %%%%%%%%%%    pl = plist('key1, val1, 'key2', 'val2' , ...)    %%%%%%%%%%
0139     params = [];
0140     argin = varargin;
0141     while length(argin) >= 2
0142         key = upper(argin{1});
0143         val =         argin{2};
0144         argin = argin(3:end);
0145
0146         if ~ischar(key)
0147             error('### the key '%s' must be a character string', key)
0148         end
0149
0150         found = 0;
0151         for ii = 1:length(params)
0152             if strcmpi(params(ii).key, key)
0153                 % If the key exist in pl then do nothing, otherwise add the key value
0154                 warning('### Do not use the same key twice. ');
0155                 found = 1;
0156             end
0157         end
0158
0159         if found == 0
0160             params = [params param(key, val)];
0161         end
0162
0163     end
0164
0165     pl = init();
0166     pl.params = params;
0167
0168 else
0169     error('### Unknown number of constructor arguments. ');
0170 end
0171
0172 end
0173
0174 % END

```


param

PURPOSE

PARAM Parameter object class constructor.

SYNOPSIS

function p = param(varargin)

DESCRIPTION

```
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 1.16 2007/12/12 18:45:08 ingo Exp $

HISTORY:     30-01-2007 M Hewitson
              Creation

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

CROSS-REFERENCE INFORMATION

This function calls:

- [param](#) PARAM Parameter object class constructor.
- [xmlparse](#) XMLPARSE creates a parameter from a parameter xml file.

This function is called by:

- **param** PARAM Parameter object class constructor.
- **save** SAVE a param object to file.

SUBFUNCTIONS

- **function p = init()**

SOURCE CODE

```

0001 function p = param(varargin)
0002 % PARAM Parameter object class constructor.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: PARAM Parameter object class constructor.
0007 %             Create a parameter object.
0008 %
0009 % CONSTRUCTOR: p = param();           - creates an empty parameter
0010 %
0011 %             p = param(pl)           - creates a parameter from a
0012 %                                     parameter list with the parameters:
0013 %                                     - 'key' and 'val', or
0014 %                                     - 'filename'
0015 %
0016 %             p = param(filename)     - load a parameter from file
0017 %
0018 %             p = param('name', val)  - creates a numeric parameter
0019 %
0020 %             p = param('name', 'val') - creates a string parameter
0021 %
0022 % VERSION:      $Id: param.m,v 1.16 2007/12/12 18:45:08 ingo Exp $
0023 %
0024 % HISTORY:      30-01-2007 M Hewitson
0025 %                Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 VERSION = '$Id: param.m,v 1.16 2007/12/12 18:45:08 ingo Exp $';
0030
0031 % Handle 'Params' call or 'Version' call
0032 if nargin == 2 && ischar(varargin{2}) && strcmp(varargin{2}, 'Params')
0033     p = plist();
0034     return
0035 elseif nargin == 2 && ischar(varargin{2}) && strcmp(varargin{2}, 'Version')
0036     p = VERSION;
0037     return
0038 end
0039
0040

```



```

0041 %%%%%%%%%%% define parameter properties %%%%%%%%%%%
%%
0042
0043 function p = init()
0044     p.name     = 'param object';
0045     p.key      = '';
0046     p.val      = [];
0047     p.created  = time;
0048     p.version  = VERSION;
0049     p = class(p, 'param');
0050 end
0051
0052 %%%%%%%%%%% Create parameter object %%%%%%%%%%%
%%
0053
0054 %%%%%%%%%%% p = param() %%%%%%%%%%%
0055 % create default parameter object
0056 if nargin == 0
0057
0058     p = init();
0059
0060 elseif nargin == 1
0061
0062     %%%%%%%%%%% Create from XML fragment %%%%%%%%%%%
0063     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0064         p = fromxml(varargin{1});
0065
0066     %%%%%%%%%%% p = param(plist) %%%%%%%%%%%
0067     elseif isa(varargin{1}, 'param')
0068         p = varargin{1};
0069
0070     %%%%%%%%%%% p = param(plist) %%%%%%%%%%%
0071     elseif isa(varargin{1}, 'plist')
0072
0073         pl = varargin{1};
0074         key = upper(find(pl, 'key'));
0075         val =         find(pl, 'val');
0076         if isempty(key)
0077             error('### building a parameter from a plist requires one parameter in
the plist is called ''key''');
0078         end
0079         if isempty(val)
0080             error('### building a parameter from a plist requires one parameter in
the plist is called ''val''');
0081         end
0082
0083         p      = init();
0084         p.key   = key;
0085         p.val   = val;
0086         p.name  = p.key;
0087
0088         %%%%%%%%%%% p = param(struct) %%%%%%%%%%%

```

```

0089 elseif isstruct(varargin{1})
0090
0091     p          = init();
0092     p.name     =     varargin{1}.name;
0093     p.key      = upper(varargin{1}.key);
0094     p.val      =     varargin{1}.val;
0095     if isstruct(varargin{1}.created)
0096         varargin{1}.created = time(varargin{1}.created);
0097     end
0098     p.version = varargin{1}.version;
0099
0100 elseif ischar(varargin{1})
0101
0102     filename = varargin{1};
0103     [path, name, ext, vers] = fileparts(filename);
0104     switch ext
0105         case '.mat'
0106             p = load(filename);
0107             p = p.a;
0108         case '.xml'
0109             p = xmlparse(param, filename);
0110         otherwise
0111             error('### Unknown file type.');
```

0112 end

0113

0114 else

0115 error('### unknown constructor type for param object.');

0116 end

0117 elseif nargin == 2

0118

0119 if isa(varargin{1}, 'database')

0120

0121 p = retrieve(varargin{1}, varargin{2:end});

0122

0123 %%%%%%%%% p = param(key, value) %%%%%%%%%

0124 else

0125 p = init();

0126 p.key = upper(varargin{1});

0127 p.val = varargin{2};

0128 p.name = p.key;

0129 end

0130 else

0131 error('### Unknown number of constructor arguments.');

0132 end

0133

0134 end % function p = param(varargin)

0135

0136

0137

0138

0139

[Home](#) > [classes](#) > [@specwin](#) > specwin.m

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

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 1.17 2008/01/11 14:05:13 ingo Exp $
```

HISTORY: 30-01-07 M Hewitson
Creation

[illegible]

CROSS-REFERENCE INFORMATION

This function calls:

- **char** CHAR convert a specwin object into a string.
- **specwin** SPECWIN spectral window object class constructor.
- **xmlparse** XMLPARSE creates a specwin from a specwin xml file.

This function is called by:

- **specwin** SPECWIN spectral window object class constructor.

SUBFUNCTIONS

- function `w = init()`

SOURCE CODE

```
0001 function w = specwin(varargin)
0002 % SPECWIN spectral window object class constructor.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SPECWIN spectral window object class constructor.
0007 %              Create a spectral window from libSpecWin.
0008 %
0009 %   PROPERTIES:
0010 %       name      - name of window object
0011 %       alpha     - alpha parameter for various window functions
0012 %       psll      - peak sidelobe level
0013 %       rov       - recommended overlap
0014 %       nenbw     - normalised equivalent noise bandwidth
0015 %       w3db      - 3 dB bandwidth in bins
0016 %       flatness  - window flatness
0017 %       ws        - sum of window values
0018 %       ws2       - sum of squares of window values
```

```

0019 %      nfft      - window length
0020 %      win       - window samples
0021 %
0022 % CONSTRUCTORS:
0023 %
0024 %      w = specwin()           - creates an empty object
0025 %      w = specwin(w)        - copies a specwin object
0026 %      w = specwin('name', N) - creates a specwin object of a
0027 %                               particular type and length.
0028 %                               (see below for supported types.)
0029 %      w = specwin('Kaiser', N, psll) - create a specwin Kaiser window
0030 %                               with the prescribed psll.
0031 %      w = specwin('Types')    - return a list of all possible
0032 %                               standard windows
0033 %
0034 % 'name' should be one of the following standard windows:
0035 %
0036 %      Rectangular, Welch, Bartlett, Hanning, Hamming,
0037 %      Nuttall3, Nuttall4, Nuttall3a, Nuttall3b, Nuttall4a
0038 %      Nuttall4b, Nuttall4c, BH92, SFT3F, SFT3M, FTNI, SFT4F, SFT5F
0039 %      SFT4M, FTHP, HFT70, FTSRS, SFT5M, HFT90D, HFT95, HFT116D
0040 %      HFT144D, HFT169D, HFT196D, HFT223D, HFT248D
0041 %
0042 % VERSION:      $Id: specwin.m,v 1.17 2008/01/11 14:05:13 ingo Exp $
0043 %
0044 % HISTORY:      30-01-07 M Hewitson
0045 %                Creation
0046 %
0047 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0048
0049 %      w = specwin('Flatop', N, psll) - create a specwin Flatop window
0050 %                               with the prescribed psll.
0051
0052 VERSION = '$Id: specwin.m,v 1.17 2008/01/11 14:05:13 ingo Exp $';
0053
0054 % Check if this is a call for parameters
0055 if nargin == 2
0056     if isa(varargin{1}, 'specwin') && ischar(varargin{2})
0057         in = char(varargin{2});
0058         if strcmp(in, 'Params')
0059             w = plist();
0060             return
0061         elseif strcmp(in, 'Version')
0062             w = VERSION;
0063             return
0064         end
0065     end
0066 end
0067
0068 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% define parameter properties list %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0069
0070 function w = init()

```

```

0071
0072     w.name      = 'no window';
0073     w.alpha     = -1;
0074     w.psll      = -1;
0075     w.rov       = -1;
0076     w.nenbw     = -1;
0077     w.w3db      = -1;
0078     w.flatness  = -1;
0079     w.ws        = -1;
0080     w.ws2       = -1;
0081     w.win       = -1;
0082     w.created   = '';
0083     w.version   = VERSION;
0084 end
0085
0086
0087
0088 %%%%%%%%%%% spw = specwin() %%%%%%%%%%%
0089 if nargin == 0
0090
0091     w = init();
0092     w = class(w, 'specwin');
0093
0094 elseif nargin == 1
0095     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0096         w = fromxml(varargin{1});
0097
0098 %%%%%%%%%%% spw = specwin(specwin) %%%%%%%%%%%
0099 elseif isa(varargin{1}, 'specwin')
0100     w = varargin{1};
0101
0102 %%%%%%%%%%% spw = specwin(struct) %%%%%%%%%%%
0103 elseif isstruct(varargin{1})
0104
0105     w = class(varargin{1}, 'specwin');
0106     return
0107
0108 %%%%%%%%%%% spw = specwin(plist) %%%%%%%%%%%
0109 elseif isa(varargin{1}, 'plist')
0110
0111     pl    = varargin{1};
0112     wname = find(pl, 'Name');
0113     N     = find(pl, 'N');
0114     psll  = find(pl, 'PSLL');
0115
0116     switch wname
0117     case 'Kaiser'
0118         w = specwin(wname, N, psll);
0119     otherwise
0120         w = specwin(wname, N);
0121 end
0122

```

```

0123 elseif ischar(varargin{1})
0124
0125     %%%%%%%%%% spw = specwin('Types') %%%%%%%%%%
0126     if strcmp(varargin{1}, 'Types')
0127
0128         % return list of windows
0129         w = {...
0130             'Rectangular', 'Welch', 'Bartlett', 'Hanning', 'Hamming', ...
0131             'Nuttall3', 'Nuttall4', 'Nuttall3a', 'Nuttall3b', 'Nuttall4a', ...
0132             'Nuttall4b', 'Nuttall4c', 'BH92', 'SFT3F', 'SFT3M', 'FTNI', 'SFT4F',
0133             'SFT5F', ...
0134             'SFT4M', 'FTHP', 'HFT70', 'FTSRS', 'SFT5M', 'HFT90D', 'HFT95',
0135             'HFT116D', ...
0136             'HFT144D', 'HFT169D', 'HFT196D', 'HFT223D', 'HFT248D' ...
0137         };
0138
0139     %%%%%%%%%% spw = specwin('foo.xml') %%%%%%%%%%
0140 else
0141     filename = varargin{1};
0142     [path, name, ext, vers] = fileparts(filename);
0143     switch ext
0144     case '.mat'
0145         w = load(filename);
0146         w = w.a;
0147     case '.xml'
0148         w = xmlparse(specwin('Hanning', 1), filename);
0149     otherwise
0150         error('### Unknown file type.');
```



```
0173 %%%%%%%%%% spw = specwin('Win_type', N) %%%%%%%%%%
0174 elseif ischar(varargin{1})
0175
0176     w = get_window(varargin{1}, varargin{2});
0177     w.created = '';
0178     w.version = VERSION;
0179     w = class(w, 'specwin');
0180
0181 else
0182     error('### unknown constructor for specwin object.')
0183 end
0184
0185 %%%%%%%%%% create a Kaiser window %%%%%%%%%%
0186 %%%%%%%%%% spw = specwin('Kaiser', N, psll) %%%%%%%%%%
0187 elseif nargin == 3
0188
0189     w = get_window(varargin{1}, varargin{2}, varargin{3});
0190     w.created = '';
0191     w.version = VERSION;
0192     w = class(w, 'specwin');
0193
0194 else
0195     error('### Unknown number of constructor arguments.');
```

```
0196 end
0197
0198 end
0199
```

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);

VERSION:      $Id: pole.m,v 1.15 2008/01/11 13:41:12 ingo Exp $

HISTORY:      02-04-2007 M Hewitson
               Creation

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

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a pole object into a string.
- [pole](#) POLE construct a pole object.
- [xmlparse](#) XMLPARSE creates a pole from a pole xml file.

This function is called by:

- [pole](#) POLE construct a pole object.

SUBFUNCTIONS

- [function p = init\(\)](#)
- [function p = poleFromPlist\(p, pl\)](#)

SOURCE CODE

```

0001 function p = pole(varargin)
0002 % POLE construct a pole object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: POLE construct a pole object.
0007 %
0008 % CONSTRUCTORS: p = pole(f);
0009 %                p = pole(f,q);
0010 %
0011 % VERSION:      $Id: pole.m,v 1.15 2008/01/11 13:41:12 ingo Exp $
0012 %
0013 % HISTORY:      02-04-2007 M Hewitson
0014 %                Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: pole.m,v 1.15 2008/01/11 13:41:12 ingo Exp $';
0019
0020 %%%% Check if this is a call for parameters %%%%
0021 if nargin == 2
0022     if isa(varargin{1}, 'pole') && ischar(varargin{2})
0023         in = char(varargin{2});
0024         if strcmp(in, 'Params')
0025             p = plist;
0026             return
0027         elseif strcmp(in, 'Version')
0028             p = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% define pole properties %%%%%%%%%
0035
0036 function p = init()
0037     p.name      = 'not defined';
0038     p.f         = NaN;
0039     p.q         = NaN;
0040     p.ri        = NaN;
0041     p.version   = VERSION;
0042     p.created   = time;
0043     p           = class(p, 'pole');
0044 end
0045
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Create pole object %%%%%%%%%
0047
0048 %%%%%%%%% p = pole() %%%%%%%%%

```

```

0049 if nargin == 0
0050
0051     p = init();
0052
0053 elseif nargin == 1
0054
0055     %%%%%%%%% Create from XML fragment %%%%%%%%%
0056     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0057         p = fromxml(varargin{1});
0058
0059     %%%%%%%%% From File %%%%%%%%%
0060     elseif ischar(varargin{1})
0061
0062         filename = varargin{1};
0063         [path, name, ext, vers] = fileparts(filename);
0064         switch ext
0065             case '.mat'
0066                 p = load(filename);
0067                 p = p.a;
0068             case '.xml'
0069                 p = xmlparse(pole, filename);
0070             otherwise
0071                 error('### Unknown file type.');
```

0072 end

0073

0074 %%%%%%%%% p = pole(number) %%%%%%%%%

0075 %%%%%%%%% p = pole(complex number) %%%%%%%%%

0076 elseif isnumeric(varargin{1})

0077

0078 p = init();

0079

0080 if isreal(varargin{1})

0081 p.name = 'real pole';

0082 p.f = varargin{1};

0083 p.q = NaN;

0084 p.ri = pfq2ri(p.f);

0085 else

0086 p.name = 'complex pole';

0087 [p.f, p.q] = pri2fq(varargin{1});

0088 p.ri = [varargin{1} conj(varargin{1})];

0089 end

0090

0091 %%%%%%%%% p = pole(struct) %%%%%%%%%

0092 elseif isstruct(varargin{1})

0093

0094 p = init();

0095

0096 fields = fieldnames(varargin{1});

0097 for ii = 1:length(fields)

0098 field = fields{ii};

0099 try

0100 p.(field) = varargin{1}.(field);

```

0101         catch
0102             error('### The field '%s' in the struct is not a pole property.',
field)
0103         end
0104     end
0105
0106     %%%%%%%%%%% p = pole(plist) %%%%%%%%%%%%%%
0107     elseif isa(varargin{1}, 'plist')
0108
0109         p = init();
0110         p = poleFromPlist(p, varargin{1});
0111
0112     %%%%%%%%%%% p = pole(pole) %%%%%%%%%%%%%%
0113     elseif isa(varargin{1}, 'pole')
0114         p = varargin{1};
0115     else
0116         error('### unknown constructor method for pole class.');

```

```

0152 version = find(pl, 'version');
0153 created = find(pl, 'created');
0154
0155 if ~isempty(f)
0156     if ~isempty(q)
0157         % complex pole
0158         if isnan(q)
0159             p.name = 'real pole';
0160         else
0161             p.name = 'complex pole';
0162         end
0163         p.f = f;
0164         p.q = q;
0165         p.ri = pfq2ri(p.f, p.q);
0166
0167     else
0168         % real pole
0169         p.name = 'real pole';
0170         p.f = f;
0171         p.q = NaN;
0172         p.ri = pfq2ri(p.f);
0173
0174     end
0175 elseif ~isempty(ri)
0176     % complex pole
0177     p.name = 'complex pole';
0178     [p.f, p.q] = pri2fq(ri);
0179     p.ri = [ri conj(ri)];
0180
0181 else
0182     error('### unknown constructor method for pole class.');
```

```

0183 end
0184
0185 % Set other the properties in the plist
0186 if ~isempty(name)
0187     p.name = name;
0188 end
0189 if ~isempty(version)
0190     p.version = version;
0191 end
0192 if ~isempty(created)
0193     p.created = created;
0194 end
0195
0196
0197 end
0198
0199 end % function p = pole(varargin)
0200
0201 % END

```


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

VERSION: \$Id: zero.m,v 1.15 2008/01/11 13:42:11 ingo Exp \$

HISTORY: 02-04-2007 Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a zero object into a string.
- [xmlparse](#) XMLPARSE creates a zero from a zero xml file.
- [zero](#) ZERO construct a pole object.

This function is called by:

- [zero](#) ZERO construct a pole object.

SUBFUNCTIONS

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

SOURCE CODE

```

0001 function p = zero(varargin)
0002 % ZERO construct a pole object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION:  ZERO construct a pole object.
0007 %
0008 % CONSTRUCTORS: z = zero(f);      - frequency (Hz)
0009 %               z = zero(f,q);    - frequency (Hz), Q
0010 %               z = zero(c);      - complex number
0011 %               z = zero(pl);     - plist()
0012 %
0013 % PARAMETERS:   'f' - frequency
0014 %               'q' - Q
0015 %               'ri' - complex value
0016 %
0017 % VERSION:      $Id: zero.m,v 1.15 2008/01/11 13:42:11 ingo Exp $
0018 %
0019 % HISTORY:      02-04-2007 Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: zero.m,v 1.15 2008/01/11 13:42:11 ingo Exp $';
0025
0026 %%%% Check if this is a call for parameters %%%%
0027 if nargin == 2
0028     if isa(varargin{1}, 'zero') && ischar(varargin{2})
0029         in = char(varargin{2});
0030         if strcmp(in, 'Params')
0031             p = plist;
0032             return
0033         elseif strcmp(in, 'Version')
0034             p = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% define zero properties %%%%%%%%%
0041
0042 function p = init()

```

```

0043     p.name      = 'not defined';
0044     p.f          = NaN;
0045     p.q          = NaN;
0046     p.ri         = NaN;
0047     p.version    = VERSION;
0048     p.created    = time;
0049     p            = class(p, 'zero');
0050 end
0051
0052 %%%%%%%%%%%%%%% Create zero object %%%%%%%%%%%%%%%
0053
0054 %%%%%%%%%%%%%%% p = zero() %%%%%%%%%%%%%%%
0055 if nargin == 0
0056
0057     p = init();
0058
0059 elseif nargin == 1
0060
0061     %%%%%%%%%%%%%%% Create from XML fragment %%%%%%%%%%%%%%%
0062     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0063         p = fromxml(varargin{1});
0064     %%%%%%%%%%%%%%% From File %%%%%%%%%%%%%%%
0065     elseif ischar(varargin{1})
0066
0067         filename = varargin{1};
0068         [path, name, ext, vers] = fileparts(filename);
0069         switch ext
0070             case '.mat'
0071                 p = load(filename);
0072                 p = p.a;
0073             case '.xml'
0074                 p = xmlparse(zero, filename);
0075             otherwise
0076                 error('### Unknown file type.');
```

```

0095     end
0096
0097     %%%%%%%%%% p = zero(struct) %%%%%%%%%%
0098     elseif isstruct(varargin{1})
0099
0100         p = init();
0101
0102         fields = fieldnames(varargin{1});
0103         for ii = 1:length(fields)
0104             field = fields{ii};
0105             try
0106                 p.(field) = varargin{1}.(field);
0107             catch
0108                 error('### The field '%s' in the struct is not a zero property.',
field)
0109             end
0110         end
0111
0112     %%%%%%%%%% p = zero(plist) %%%%%%%%%%
0113     elseif isa(varargin{1}, 'plist')
0114
0115         p = init();
0116         p = zeroFromPlist(p, varargin{1});
0117
0118     %%%%%%%%%% p = zero(zero) %%%%%%%%%%
0119     elseif isa(varargin{1}, 'zero')
0120         p = varargin{1};
0121
0122     else
0123         error('### unknown constructor method for zero class.');
```

```

0124     end
0125
0126 elseif nargin == 2
0127     %%%%%%%%%% From DATABASE
0128     if isa(varargin{1}, 'database')
0129         p = retrieve(varargin{1}, varargin{2:end});
0130     %%%%%%%%%% p = zero(f, q) %%%%%%%%%%
0131     elseif isnumeric(varargin{1}) && isnumeric(varargin{2})
0132
0133         p = init();
0134         if isnan(varargin{2})
0135             p.name = 'real zero';
0136         else
0137             p.name = 'complex zero';
0138         end
0139         p.f = varargin{1};
0140         p.q = varargin{2};
0141         p.ri = pfq2ri(p.f, p.q);
0142
0143     else
0144         error('### unknown constructor method for zero class.');
```

```

0145     end

```

```

0146
0147 end
0148
0149 %-----
0150 % construct a zero from plist
0151 %
0152 function p = zeroFromPlist(p, pl)
0153
0154     f = find(pl, 'f');
0155     q = find(pl, 'q');
0156     ri = find(pl, 'ri');
0157
0158     name = find(pl, 'name');
0159     version = find(pl, 'version');
0160     created = find(pl, 'created');
0161
0162     if ~isempty(f)
0163         if ~isempty(q)
0164             % complex zero
0165             if isnan(q)
0166                 p.name = 'real zero';
0167             else
0168                 p.name = 'complex zero';
0169             end
0170             p.f = f;
0171             p.q = q;
0172             p.ri = pfq2ri(p.f, p.q);
0173
0174         else
0175             % real zero
0176             p.name = 'real zero';
0177             p.f = f;
0178             p.q = NaN;
0179             p.ri = pfq2ri(p.f);
0180
0181         end
0182     elseif ~isempty(ri)
0183         % complex zero
0184         p.name = 'complex zero';
0185         [p.f, p.q] = pri2fq(ri);
0186         p.ri = [ri conj(ri)];
0187
0188     else
0189         error('### unknown constructor method for zero class.');
```

```

0190     end
0191
0192     % Set other the properties in the plist
0193     if ~isempty(name)
0194         p.name = name;
0195     end
0196     if ~isempty(version)
0197         p.version = version;

```

```
0198     end
0199     if ~isempty(created)
0200         p.created = created;
0201     end
0202
0203 end
0204
0205
0206 end % function p = zero(varargin)
0207
```

[Home](#) > [classes](#) > [@pzmodel](#) > pzmodel.m

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')
 pzm = pzmodel(pl); % construct from plist

PARAMETERS: 'gain' - model gain
 'poles' - vector of pole objects
 'zeros' - vector of zero objects
 'name' - name of model

VERSION: \$Id: pzmodel.m,v 1.11 2008/01/07 18:13:10 ingo Exp \$

HISTORY: 03-04-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a pzmodel object into a string.
- [pzmodel](#) PZMODEL constructor for pzmodel class.
- [xmlparse](#) XMLPARSE creates a pzmodel from a pzmodel xml file.

This function is called by:

- [pzmodel](#) PZMODEL constructor for pzmodel class.

SUBFUNCTIONS

- `function pz = init()`

SOURCE CODE

```

0001 function pzm = pzmodel(varargin)
0002 % PZMODEL constructor for pzmodel class.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: PZMODEL constructor for pzmodel class.
0007 %
0008 % CONSTRUCTOR: pzm = pzmodel(g, p, z) % construct from gain, poles, zeros
0009 %                pzm = pzmodel(g, p, z, 'name')
0010 %                pzm = pzmodel(pl);      % construct from plist
0011 %
0012 % PARAMETERS:  'gain' - model gain
0013 %                'poles' - vector of pole objects
0014 %                'zeros' - vector of zero objects
0015 %                'name' - name of model
0016 %
0017 % VERSION:      $Id: pzmodel.m,v 1.11 2008/01/07 18:13:10 ingo Exp $
0018 %
0019 % HISTORY:      03-04-2007 M Hewitson
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: pzmodel.m,v 1.11 2008/01/07 18:13:10 ingo Exp $';
0025
0026 %%%% Check if this is a call for parameters %%%%
0027 if nargin == 2
0028     if isa(varargin{1}, 'pzmodel') && ischar(varargin{2})
0029         in = char(varargin{2});
0030         if strcmp(in, 'Params')
0031             pzm = plist;
0032             return
0033         elseif strcmp(in, 'Version')
0034             pzm = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% define pzmodel properties %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0041
0042 function pz = init()
0043     pz.name = 'pzmodel object';
0044     pz.gain = 0;

```

```

0045     pz.poles    = pole;
0046     pz.zeros    = zero;
0047     pz.created  = time;
0048     pz.version  = VERSION;
0049     pz          = class(pz, 'pzmodel');
0050 end
0051
0052 %%%%%%%%%%%%%%% Create pzmodel object %%%%%%%%%%%%%%%
0053
0054 %%%%%%%%%%%%%%% pzm = pzmodel() %%%%%%%%%%%%%%%
0055 if nargin == 0
0056
0057     pzm = init();
0058
0059 elseif nargin == 1
0060
0061     %%%%%%%%%%%%%%% Create from XML fragment %%%%%%%%%%%%%%%
0062     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0063         pzm = fromxml(varargin{1});
0064
0065     %%%%%%%%%%%%%%% p = pzmodel(pzmodel) %%%%%%%%%%%%%%%
0066     elseif isa(varargin{1}, 'pzmodel')
0067         pzm = varargin{1};
0068
0069     %%%%%%%%%%%%%%% From File %%%%%%%%%%%%%%%
0070     elseif ischar(varargin{1})
0071
0072         filename = varargin{1};
0073         [path, name, ext, vers] = fileparts(filename);
0074         switch ext
0075             case '.mat'
0076                 pzm = load(filename);
0077                 pzm = pzm.a;
0078             case '.xml'
0079                 pzm = xmlparse(pzmodel, filename);
0080             otherwise
0081                 error('### Unknown file type.');
```



```

0097     poles = [];
0098     for jj = 1:length(ps)
0099         if isstruct(ps(jj))
0100             poles = [poles pole(ps(jj))];
0101         else
0102             poles = [poles ps(jj)];
0103         end
0104     end
0105     pzm.poles = poles;
0106
0107     %%% zero -> pole-object
0108     elseif strcmp(field, 'zeros')
0109
0110         zs = varargin{1}.zeros;
0111         zeros = [];
0112         for jj = 1:length(zs)
0113             if isstruct(zs(jj))
0114                 zeros = [zeros zero(zs(jj))];
0115             else
0116                 zeros = [zeros zs(jj)];
0117             end
0118         end
0119         pzm.zeros = zeros;
0120
0121     %%% created -> time-object
0122     elseif strcmp(field, 'created')
0123         if isstruct(varargin{1}.created)
0124             pzm.created = time(varargin{1}.created);
0125         else
0126             pzm.created = varargin{1}.created;
0127         end
0128     %%% All other
0129     else
0130         try
0131             pzm.(field) = varargin{1}.(field);
0132         catch
0133             error('### The field '%s' in the struct is not a pole property.',
field)
0134         end
0135     end
0136 end
0137
0138 %%%%%%%%%%% pzm = pzmodel(plist) %%%%%%%%%%%
0139 else
0140
0141     pzm = init();
0142     pl = varargin{1};
0143
0144     pzm.gain    = find(pl, 'gain');
0145     pzm.poles   = find(pl, 'poles');
0146     pzm.zeros   = find(pl, 'zeros');
0147
0148     % If the following properties are not set in the plist then use the default

```

```

0149 % values
0150 if ~isempty(find(pl, 'version'))
0151     pzm.version = find(pl, 'version');
0152 end
0153 if ~isempty(find(pl, 'created'))
0154     pzm.created = find(pl, 'created');
0155 end
0156 if ~isempty(find(pl, 'name'))
0157     pzm.name = find(pl, 'name');
0158 end
0159
0160 end
0161
0162 elseif nargin == 2
0163     %%%%%%%%%%% From DATABASE %%%%%%%%%%%
0164     if isa(varargin{1}, 'database')
0165         pzm = retrieve(varargin{1}, varargin{2:end});
0166     else
0167         error('### incorrect pzmodel constructor.');

```

```
0202 end
0203
0204 end
0205
0206 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

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.

 [aosplit](#)

SPLIT an analysis object into the specified segments.

 [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.

 [decimate](#)

DECIMATE AOs containing time-series data.

 [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.

 [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.
 fft	FFT overloads the fft operator for Analysis objects.
 filter	FILTER overrides the filter function for analysis objects.
 filtfilt	FILTFILT overrides the filtfilt function for analysis objects.
 find	FIND particular samples that satisfy the input query and return a new AO.
 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	I PLOT 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 concatenate 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.
 plot_old	PLOT plots an array 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.
 simple_plot	PLOT a simple plot of analysis objects.
 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).
 submit_fastinsert	SUBMIT AO(s) to the MySQL server.
 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.
 testAO	TESTAO checks that an analysis object has the correct class fields.
 times	TIMES overloads .* operator for analysis objects.
 transpose	TRANPOSE 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.
 xml	XML convert an analysis object to an DOM XML object.
 xmladd	XMLADD add an analysis object to the given node of the input XML DOM
 xmlparse	XMLPARSE creates a ao from a ao xml file.

Subsequent directories:

- [CVS](#)
- [private](#)
- [testing](#)

Matlab Index

Matlab Directories

- [classes/@ao](#)
- [classes/@ao/private](#)
- [classes/@ao/testing](#)
- [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/analyses/stuttgart](#)
- [m/etc](#)
- [m/gui/ao_browser](#)
- [m/gui/ltpdaRepoGUI](#)
- [m/gui/ltpdaRepoGUI/callbacks](#)
- [m/gui/pzmodel_designer](#)
- [m/helper](#)
- [m/math](#)
- [m/mysql](#)
- [m/noisegenerator](#)
- [m/noisegenerator/private](#)
- [m/plottools](#)
- [m/sigproc/frequency_domain](#)
- [m/sigproc/time_domain](#)
- [m/timetools](#)
- [mex](#)
- [mex/macintel/2007b](#)
- [mex/windows/2007b](#)

Matlab Files found in these Directories

Contents	find_real_ip	ltpda_xspec	sin
Contents	format	ltpda_xticks	single_operation
Contents	fq2fac	ltpda_yaxis	single_operation
Contents	fromxml	ltpda_yscale	single_operation
Contents	fromxml	ltpda_yticks	single_operation
Contents	fromxml	ltpda_zaxis	single_operation
Contents	fromxml	ltpda_zscale	spectrogram
Contents	fromxml	md5	specwin
Contents	fromxml	mean	split
Contents	fromxml	median	sqlResultsGUI
Contents	fromxml	mfiltload	sqrt
Contents	fromxml	mfir	stairs
Contents	fromxml	miir	std

Contents	fromxml	minus	string
Contents	fromxml	minus	string
Contents	fromxml	mk_test_objs	string
Contents	fromxml	mkbandpass	string
Contents	fromxml	mkbandpass	string
Contents	fromxml	mkbandreject	string
Contents	fromxml	mkbandreject	string
Contents	fromxml	mkhighpass	string
Contents	fsdata	mkhighpass	string
Contents	gcbsh	mklowpass	string
Contents	ge	mklowpass	strpad
Contents	get	mpower	submit_fastinsert
Contents	get	mrdivide	subsasgn
Contents	get	mtimes	subsasgn
Contents	get	muitable	subsasgn
Contents	get	mux	subsasgn
Contents	get	mux	subsasgn
Contents	get	mysql_connect	subsasgn
Contents	get	mysql_getMD5hash	subsasgn
Contents	get	mysql_getMaxId	subsasgn
Contents	get	mysql_getObjIds	subsasgn
Contents	get	mysql_getObjType	subsasgn
Contents	get	mysql_getUserID	subsasgn
Contents	get	mysql_getXdoc	subsasgn
Contents	get	ne	subsasgn
Contents	get	ne	subsasgn
Contents	get	ne	subsasgn
Contents	get	ne	subsasgn
Contents	getFields	ne	subsasgn
Contents	getNodes	ne	subsasgn
Contents	getNodes_plot	ne	subsasgn
Contents	getTables	ne	subsref
Contents	getWorkspaceObjs	ne	subsref

Contents	get_curr_m_file_path	ne	subsref
Contents	get_window	ne	subsref
Contents	get_xy_values	ne	subsref
Contents	get_xy_values	ne	subsref
Contents	get_xy_values	ne	subsref
Contents	get_xy_values	ne	subsref
abccascade	get_xyz_values	ne	subsref
abs	getlowerFreq	ne	subsref
acos	getupperFreq	ne	subsref
ao	gt	ne	subsref
ao2m	hist	ngconv	subsref
ao_fcn_template	hist2dot	nginit	subsref
aooperate	hist2m	ngprop	subsref
aosplit	history	ngsetup	subsref
append	horzcat	norm	subsref
asin	imag	nparams	subsref
atan	index	param	subsref
attachm	interp	parseFilterParams	sum
attachmdl	inv	parseFilterParams	svd
buildConditions	iplot	pfq2ri	tan
buildquery	isfield	pfq2ri	testAO
cat	isfield	phase	test_dkalman
cat	isfield	phase	time
cat	isfield	plist	timeformat
cat	isfield	plot	times
cb_classMethodList	isfield	plot	timespan
cb_condPlusBtn	isfield	plot	tomfir
cb_condSubtractBtn	isfield	plot_old	tomiir
cb_connectBtn	isfield	plus	transpose
cb_dbSelect	isfield	plus	tsdata
cb_executeQuery	isfield	pole	uitabdemo
cb_fieldSelect	isfield	polydetrend	uitabpanel
cb_getDBs	isfield	polyfit	upsample

cb_mainfigClose	isfield	power	var
cb_refreshObjsList	isfield	pri2fq	vertcat
cb_refreshObjsListQL	isfield	pri2fq	win_BH92
cb_retrieveBtn	isfield	provenance	win_Bartlett
cb_serverSelect	isfield	pset	win_FTHP
cb_submitBtn	isfield	pzm2ab	win_F TNI
cb_tableSelect	islinespec	pzmodel	win_FTSRS
cb_workspaceObjsList	isparam	pzmodel_helper	win_HFT116D
cb_workspaceObjsListQL	isvalid	pzmresp	win_HFT144D
cdata	join	rdivide	win_HFT169D
char	kaiser_alpha	real	win_HFT196D
char	kaiser_flatness	redesign	win_HFT223D
char	kaiser_nenbw	redesign	win_HFT248D
char	kaiser_rov	remove	win_HFT70
char	kaiser_w3db	repogui	win_HFT90D
char	le	resample	win_HFT95
char	len	reshapeF	win_Hamming
char	ln	reshapeT	win_Hanning
char	log	reshapeX	win_Nuttall3
char	log10	resp	win_Nuttall3a
char	logindlg	resp	win_Nuttall3b
char	lt	resp	win_Nuttall4
char	ltpda_allgrid	resp	win_Nuttall4a
char	ltpda_allxaxis	resp	win_Nuttall4b
char	ltpda_allxlabel	resp	win_Nuttall4c
char	ltpda_allxscale	retrieve	win_Rectangular
char	ltpda_allyaxis	retrieve	win_SFT3F
char	ltpda_allylabel	retrieve	win_SFT3M
collect_inputs	ltpda_allyscale	retrieve	win_SFT4F
combine	ltpda_chkts	retrieve	win_SFT4M
complex	ltpda_cohere	retrieve	win_SFT5F
compute_interval	ltpda_compute_f	retrieve	win_SFT5M
conj	ltpda_cpsd	retrieve	win_Welch

conv_noisegen	ltpda_cscale	retrieve	wrapstring
cos	ltpda_dbquery	retrieve	xdoc2ao
cp2iir	ltpda_deg2rad	retrieve	xml
ctranspose	ltpda_dft	retrieve	xml
cz2iir	ltpda_dft	retrieve	xml
decimate	ltpda_dft	retrieve	xml
demux	ltpda_diff_acc	retrieve	xml
det	ltpda_dkalman	retrieve	xml
diag	ltpda_fields2list	retrieve	xml
display	ltpda_filescan	retrieve	xml
display	ltpda_fq2ri	rp2iir	xml
display	ltpda_fq2ri2	rstruct	xml
display	ltpda_funchash	rz2iir	xml
display	ltpda_getmax	save	xml
display	ltpda_gps2utc	save	xml
display	ltpda_hash	save	xml
display	ltpda_insert	save	xml
display	ltpda_io_collect	save	xml
display	ltpda_isobject	save	xml
display	ltpda_isuserobject	save	xml
display	ltpda_javacomponent	save	xmladd
display	ltpda_label	save	xmladd
display	ltpda_lcohere	save	xmladd
display	ltpda_lcpsd	save	xmladd
display	ltpda_legendAdd	save	xmladd
display	ltpda_lincom	save	xmladd
display	ltpda_linedetect	save	xmladd
display	ltpda_lpsd	save	xmladd
dkalman_matrices	ltpda_ltf	save	xmladd
dunzip	ltpda_lxspec	save	xmladd
dzip	ltpda_mat2str	save	xmladd
e2e2ao	ltpda_matchvectors	save	xmladd
eig	ltpda_msuptitle	select	xmladd

eq	ltpda_nfest	set	xmladd
eq	ltpda_noisegen	set	xmladd
eq	ltpda_num2str	set	xmladd
eq	ltpda_obj_retrieve	set	xmladd
eq	ltpda_obj_submit	set	xmladd
eq	ltpda_phase	set	xmlparse
eq	ltpda_polydetrend	set	xmlparse
eq	ltpda_psd	set	xmlparse
eq	ltpda_pwelch	set	xmlparse
eq	ltpda_rand	set	xmlparse
eq	ltpda_reformat_date	set	xmlparse
eq	ltpda_ri2fq	set	xmlparse
eq	ltpda_rms	set	xmlparse
eq	ltpda_saveobj	set	xmlparse
eq	ltpda_startup	set	xmlparse
eq	ltpda_str2cells	set	xmlparse
eq	ltpda_strs2cells	set	xmlparse
eq	ltpda_structcat	set	xmlparse
eq	ltpda_structidx	set	xmlparse
exp	ltpda_tfe	setFreq	xmlparse
explore_ao	ltpda_timedomainfit	setTime	xmlparse
export	ltpda_uitree	setWorkspaceObjsList	xmlparse
fft	ltpda_uitreenode	setWorkspaceObjsListQL	xmlparse
fillClassMethodsList	ltpda_unwrapdeg	set_xy_axis	xydata
filload	ltpda_utc2gps	set_xy_axis	xyzdata
filter	ltpda_xaxis	set_xy_axis	zero
filtfilt	ltpda_xcorr	set_xy_axis	
find	ltpda_xml_read_datasamples	setnh	
find	ltpda_xscale	simple_plot	

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 1.8 2007/11/26 14:48:13 ingo Exp \$

HISTORY: 15-02-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR overloads char() function for analysis objects.

This function is called by:

- [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.
- **decimate** DECIMATE AOs containing time-series data.
- **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.
- **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.
- **fft** FFT overloads the fft operator for Analysis objects.
- **filter** FILTER overrides the filter function for analysis objects.
- **filtfilt** FILTFILT overrides the filtfilt function for analysis objects.
- **find** FIND particular samples that satisfy the input query and return a new AO.
- **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.
- **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.
- **plot_old** PLOT plots an array 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.
- **simple_plot** PLOT a simple plot of analysis objects.
- **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.
- **submit_fastinsert** SUBMIT AO(s) to the MySQL server.
- **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.
- **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.
- **xmlparse** XMLPARSE creates a ao from a ao xml file.

SOURCE CODE

```

0001 function varargout = char(varargin)
0002 % CHAR overloads char() function for analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION:      CHAR overloads char() function for analysis objects.
0007 %
0008 % CALL:             c = char(ao_in);
0009 %
0010 % FORMAT:           c = 'ao.name/ao.data.name class(ao.data) [size(ao.data.
yaxis)]'
0011 %

```

```

0012 % POSSIBLE VALUES: ao_in = [ao2 ao3]
0013 %                        ao_in = ao_vector
0014 %                        ao_in = ao_matrix
0015 %
0016 % VERSION:                $Id: char.m,v 1.8 2007/11/26 14:48:13 ingo Exp $
0017 %
0018 % HISTORY: 15-02-07  M Hewitson
0019 %                Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 VERSION = '$Id: char.m,v 1.8 2007/11/26 14:48:13 ingo Exp $';
0024
0025 %% Check if this is a call for parameters
0026 if nargin == 2
0027     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0028         in = char(varargin{2});
0029         if strcmp(in, 'Params')
0030             varargout{1} = plist();
0031             return
0032         elseif strcmp(in, 'Version')
0033             varargout{1} = VERSION;
0034             return
0035         end
0036     end
0037 end
0038
0039
0040 %% Collect input ao's, plist's and ao variable names
0041 in_names = {};
0042 for ii = 1:nargin
0043     in_names{end+1} = inputname(ii);
0044 end
0045
0046 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0047
0048 if ~isempty (pl)
0049     pl = combine(pl);
0050 end
0051
0052 %% go through analysis objects
0053 for j=1: numel(ao_set)
0054
0055     a = ao_set(j);
0056
0057     pstr = [];
0058     pstr = [pstr a.name];
0059
0060     % examine data object
0061     d = a.data;
0062
0063     % get data type

```

```
0064     if isempty(d)
0065         pstr = [pstr ' / No data-object'];
0066     else
0067         pstr = [pstr '/' d.name ' ' class(d)];
0068
0069         [x,y] = get_xy_values(a.data);
0070
0071         xinfo = whos('y');
0072         pstr = [pstr ' ' sprintf('[%d %d]', xinfo.size(1), xinfo.size(2)) ];
0073
0074     end
0075     % get data time
0076
0077     pstr = strrep(pstr, '_', '\_');
0078
0079     varargout{j} = pstr;
0080
0081 end
0082
0083 % END
```

[Home](#) > [classes](#) > [@ao](#) > find.m

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 1.10 2008/01/20 12:53:29 hewitson Exp \$

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

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

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.
- **aosplit** SPLIT an analysis object into the specified segments.
- **decimate** DECIMATE AOs containing time-series data.
- **filter** FILTER overrides the filter function for analysis objects.
- **filtfilt** FILTFILT overrides the filtfilt function for analysis objects.
- **find** FIND particular samples that satisfy the input query and return a new AO.
- **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.
- **ipplot** 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.
- **var** VAR overloads the var operator for Analysis objects. Compute the variance.

SUBFUNCTIONS

- **function pl_default = getDefaultPL()**

SOURCE CODE

```

0001 function bo = find(varargin)
0002 % FIND particular samples that satisfy the input query and return a new AO.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: FIND particular samples that satisfy the input query and return
a new AO.
0007 %
0008 % CALL:          b = find(a, 'x > 10 & x < 100')
0009 %              b = find(a, plist)
0010 %
0011 % PARAMETERS:   query - a string specifying a query on 'x' or 'y' for axes data
0012 %              or 'vals' for cdata
0013 %
0014 % VERSION:      $Id: find.m,v 1.10 2008/01/20 12:53:29 hewitson Exp $

```

```

0015 %
0016 % The following call returns a parameter list object that contains the
0017 % default parameter values:
0018 %
0019 % >> pl = find(ao, 'Params')
0020 %
0021 % HISTORY: 14-05-07 M Hewitson
0022 %             Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 VERSION = '$Id: find.m,v 1.10 2008/01/20 12:53:29 hewitson Exp $';
0027 ALGONAME = mfilename;
0028 queries = [];
0029
0030 %% Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             bo = getDefaultPL();
0036             return
0037         elseif strcmp(in, 'Version')
0038             bo = VERSION;
0039             return
0040         end
0041     end
0042 end
0043
0044 % Collect input ao's, plist's and ao variable names
0045 in_names = {};
0046 for ii = 1:nargin
0047     in_names{end+1} = inputname(ii);
0048
0049     if ischar(varargin{ii})
0050         queries = [queries ' & ' varargin{ii}];
0051     end
0052 end
0053
0054 [as, ps, invars] = collect_inputs(varargin, in_names);
0055
0056 % Check inputs
0057 if isempty(as)
0058     error('### Please input at least one A0.');
```

```

0067 else
0068     pl = plist();
0069 end
0070
0071 % Get sample selection from plist
0072 queries = [queries ' & ' find(pl, 'query')];
0073
0074 % We may have pre or trailing '&' now
0075 queries = strtrim(queries);
0076 if queries(1) == '&'
0077     queries = queries(2:end);
0078 end
0079 if queries(end) == '&'
0080     queries = queries(1:end-1);
0081 end
0082
0083
0084 % disp(sprintf('*** processing %s', queries));
0085
0086 % Loop over input AOs
0087 bo = [];
0088 for j=1:numel(as)
0089
0090     a = as(j);
0091     d = a.data;
0092
0093     [x,y] = get_xy_values(d);
0094
0095     % special case for the cdata class.
0096     % The user can also use the query to 'vals'
0097     vals = y;
0098
0099     cmd = sprintf('idx = find(%s);', queries);
0100     try
0101         eval(cmd);
0102     catch
0103         l_error = lasterror;
0104         error('%s\n\n### please use x or y for the queries.', l_error.message);
0105     end
0106
0107     if isa(d, 'cdata')
0108         % Have each input of 'vals' a 'tags'
0109         if length(x) == length(y)
0110             d = set_xy_axis(d, x(idx), y(idx));
0111         else
0112             d = set_xy_axis(d, x, y(idx));
0113         end
0114     else
0115         d = set_xy_axis(d, x(idx), y(idx));
0116     end
0117
0118     % Make output AO

```

```
0119
0120 % create new output history
0121 h = history(ALGONAME, VERSION, plist(param('queries', queries)), a.hist);
0122 h = set(h, 'invars', invars(j));
0123
0124 % make output analysis object
0125 b = ao(d, h);
0126
0127 % set name
0128 b = setnh(b, 'name', sprintf('select(%s)', invars{j}));
0129
0130 % Add to output array
0131 bo = [bo b];
0132
0133 end
0134
0135 % Reshape the ouput to the same size of the input
0136 bo = reshape(bo, size(as));
0137
0138
0139 %% Get default params
0140 function pl_default = getDefaultPL()
0141
0142     pl_default = plist(param('query', ''));
0143
0144
0145
0146 % END
```


[Home](#) > [classes](#) > [@ao](#) > hist.m

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 1.9 2008/01/20 12:53:29 hewitson Exp \$

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

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

HISTORY: 24-05-07 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.
- **hist** HIST overloads the histogram function (hist) 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:

- **ao** AO analysis object class constructor.
- **display** DISPLAY implement terminal display for analysis object.
- **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).

SUBFUNCTIONS

- **function plo = getDefaultPL()**

SOURCE CODE

```

0001 function varargout = hist(varargin)
0002 % HIST overloads the histogram function (hist) of MATLAB for Analysis Objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: HIST overloads the histogram function (hist) of MATLAB for
0007 %               Analysis Objects.
0008 %
0009 % CALL:         b = hist(a)
0010 %               b = hist(a, N)
0011 %               b = hist(a, X)
0012 %               b = hist(a, pl)
0013 %
0014 % INPUTS:       a - input analysis object(s)
0015 %               N - number of bins
0016 %               X - specify centers of bins
0017 %               pl - a parameter list
0018 %
0019 % OUTPUTS:       b - xydata type analysis object(s) containing the
0020 %                   histogrammed data
0021 %
0022 % PARAMETERS:   'N' - number of bins
0023 %               'X' - set of bin centers
0024 %

```

```

0025 % VERSION:      $Id: hist.m,v 1.9 2008/01/20 12:53:29 hewitson Exp $
0026 %
0027 % The following call returns a parameter list object that contains the
0028 % default parameter values:
0029 %
0030 % >> pl = hist(ao, 'Params')
0031 %
0032 % HISTORY: 24-05-07 M Hewitson
0033 %           Creation
0034 %
0035 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0036
0037 ALGONAME = mfilename;
0038 VERSION  = '$Id: hist.m,v 1.9 2008/01/20 12:53:29 hewitson Exp $';
0039
0040 %% Check if this is a call for parameters
0041 if nargin == 2
0042     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0043         in = char(varargin{2});
0044         if strcmp(in, 'Params')
0045             varargout{1} = getDefaultPL();
0046             return
0047         elseif strcmp(in, 'Version')
0048             varargout{1} = VERSION;
0049             return
0050         end
0051     end
0052 end
0053
0054 %----- capture input variable names, AOs, and plists
0055 Nbins = -1;
0056 Xbins = [];
0057
0058 % Collect input ao's, plist's and ao variable names
0059 in_names = {};
0060 for ii = 1:nargin
0061     in_names{end+1} = inputname(ii);
0062
0063     % Collect Xbins and Nbins
0064     if isnumeric(varargin{ii})
0065         if length(varargin{ii}) > 1
0066             Xbins = varargin{ii};
0067         else
0068             Nbins = varargin{ii};
0069         end
0070     end
0071 end
0072
0073 [as, ps, invars] = collect_inputs(varargin, in_names);
0074
0075 % produce one plist
0076 if isa(ps, 'plist')

```

```

0077     pl = combine(ps, getDefaultPL());
0078 else
0079     pl = getDefaultPL();
0080 end
0081
0082 % If we have X from command input, override plist
0083 if ~isempty(Xbins)
0084     pl = pset(pl, 'X', Xbins);
0085 end
0086
0087 % If we have N from command input, override plist
0088 if Nbins>0
0089     pl = pset(pl, 'N', Nbins);
0090 end
0091
0092 % Get parameters
0093 N = find(pl, 'N');
0094 X = find(pl, 'X');
0095
0096 %----- Loop over input AOs
0097
0098 % Initialise output
0099 bs = [];
0100
0101 % start looping
0102 for j=1:numel(as)
0103     a = as(j);
0104
0105     % get data
0106     [x, y] = get_xy_values(a.data);
0107
0108     % Histogram this data
0109     if isempty(X)
0110         %disp(sprintf('** sorting data into %d bins', N));
0111         [n,x] = hist(y, N);
0112     else
0113         %disp(sprintf('** sorting data into %d bins', length(X)));
0114         [n,x] = hist(y, X);
0115     end
0116
0117     %----- Build output AOs
0118
0119     % make a new xydata object
0120     xy = xydata(x, n);
0121     xy = set(xy, 'name', sprintf('hist(%s)', a.data.name));
0122     xy = set(xy, 'xunits', a.data.yunits);
0123     xy = set(xy, 'yunits', 'Count');
0124
0125     % make a new history object
0126     h = history(ALGONAME, VERSION, pl, a.hist);
0127     h = set(h, 'invars', invars(j));

```

```
0128
0129 % make output analysis object
0130 b = ao(xy, h);
0131
0132 % name for this object
0133 b = setnh(b, 'name', sprintf('hist(%s)', invars{j}));
0134 bs = [bs b];
0135
0136 end % end of AO loop
0137
0138 % Reshape the output to the same size of the input
0139 bs = reshape(bs, size(as));
0140
0141
0142 %----- Set outputs
0143 varargout{1} = bs;
0144
0145 %-----
0146 %-----
0147 %-----
0148 %-----
0149
0150 %-----
0151
0152 %% Get default params
0153 function plo = getDefaultPL()
0154
0155 plo = plist();
0156 plo = append(plo, param('N', 10));
0157 plo = append(plo, param('X', []));
0158
0159 % END
```

[Home](#) > [classes](#) > [@ao](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.8 2007/12/19 19:50:54 hewitson 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

```

0001 function varargout = isfield(varargin)
0002
0003 % ISFIELD tests if the given field is one of the object properties.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0008 %               properties.
0009 %
0010 % CALL:         isfield(a, 'field');
0011 %
0012 %               The following call returns a parameter list object that
0013 %               contains the default parameter values:
0014 %               >> pl = isfield(ao, 'Params')
0015 %
0016 %               The following call returns the cvs version number:
0017 %               >> ver = isfield(ao, 'Version')
0018 %
0019 % VERSION:      $Id: isfield.m,v 1.8 2007/12/19 19:50:54 hewitson Exp $
0020 %
0021 % HISTORY: 25-09-07 M Hewitson
0022 %           Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 VERSION = '$Id: isfield.m,v 1.8 2007/12/19 19:50:54 hewitson Exp $';
0027
0028 % 'Params' Call
0029 if nargin == 2
0030     if isa(varargin{1}, 'ao') && strcmp(varargin{2}, 'Params')
0031         varargout{1} = plist();
0032         return
0033     elseif isa(varargin{1}, 'ao') && strcmp(varargin{2}, 'Version')
0034         varargout{1} = VERSION;
0035         return
0036     end
0037 end
0038
0039 as = [];
0040 field = '';
0041 for j=1:nargin
0042     if isa(varargin{j}, 'ao')
0043         as = [as varargin{j}];
0044     end
0045     if ischar(varargin{j})
0046         field = varargin{j};
0047     end
0048 end
0049
0050 res = zeros(size(as));
0051 for j=1: numel(as)
0052     try

```

```
0053     t = as(j).(field);
0054     res(j) = 1;
0055     catch
0056         res(j) = 0;
0057     end
0058 end
0059
0060 varargout{1} = res;
0061
```


[Home](#) > [classes](#) > [@ao](#) > set.m

set

PURPOSE

SET set an analysis object property.

SYNOPSIS

function ao = set(ao, 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, '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 = ao(ao, 'Params')
```

VERSION: \$Id: set.m,v 1.22 2007/11/26 14:48:13 ingo Exp \$

HISTORY: 30-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [ao](#) AO analysis object class constructor.
- [char](#) CHAR overloads char() function for analysis objects.
- [set](#) SET set an analysis object property.

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.
- [aosplit](#) SPLIT an analysis object into the specified segments.
- [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.
- [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.
- [decimate](#) DECIMATE AOs containing time-series data.
- [det](#) DET overloads the determinant function for Analysis objects.
- [diag](#) DIAG overloads the diagonal operator for Analysis Objects.
- [eig](#) EIG overloads the determinant function for Analysis objects.
- [exp](#) EXP overloads the exp operator for Analysis objects. Exponential.
- [fft](#) FFT overloads the fft operator for Analysis objects.
- [filter](#) FILTER overrides the filter function for analysis objects.
- [filtfilt](#) FILTFILT overrides the filtfilt function for analysis objects.
- [find](#) FIND particular samples that satisfy the input query and return a new AO.
- [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.
- [ipplot](#) IPLOT provides an intelligent plotting tool for LTPDA.
- [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.

- **plot** PLOT a simple plot of analysis objects.
- **plot_old** PLOT plots an array 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.
- **simple_plot** PLOT a simple plot of analysis objects.
- **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.
- **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 plo = getDefaultPL()**

SOURCE CODE

```

0001 function ao = set(ao, varargin)
0002 % SET set an analysis object property.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET set an analysis object property.
0007 %
0008 % CALL: ao = set(ao, 'name',      'name');
0009 %       ao = set(ao, 'provenance', provenance-object);
0010 %       ao = set(ao, 'description','description');
0011 %       ao = set(ao, 'hist',      history-object);
0012 %       ao = set(ao, 'data',      data);
0013 %       ao = set(ao, 'mfile',      mfile);
0014 %       ao = set(ao, 'mfilename', 'mfilename');

```

```

0015 %      ao = set(ao, 'mdlfile',      'mdlfile');
0016 %      ao = set(ao, 'mdlfilename','mdlfilename');
0017 %      ao = set(ao, 'version',      'cvs-version');
0018 %      ao = set(ao, 'created',      time-object);
0019 %      ao = set(ao,  plist-object);          --> plist-object with key/value
paies
0020 %      ao = set(ao, 'fs',           fs);          --> ao.data.fs = fs;
0021 %      ao = set(ao, 't0',           t0);          --> ao.data.t0 = t0;
0022 %      ao = set(ao, 'xunits',       'xunits'); --> ao.data.xunits = 'xunits';
0023 %      ao = set(ao, 'yunits',       'yunits'); --> ao.data.xunits = 'yunits';
0024 %
0025 % The following call returns a parameter list object that contains the
0026 % default parameter values:
0027 %
0028 % >> pl = ao(ao, 'Params')
0029 %
0030 % VERSION: $Id: set.m,v 1.22 2007/11/26 14:48:13 ingo Exp $
0031 %
0032 % HISTORY: 30-01-07 M Hewitson
0033 %           Creation
0034 %
0035 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0036
0037 VERSION = '$Id: set.m,v 1.22 2007/11/26 14:48:13 ingo Exp $';
0038
0039 % Check if this is a call for parameters or for the cvs-version number
0040 if nargin == 2
0041     if isa(ao, 'ao') && ischar(varargin{1})
0042         in = char(varargin{1});
0043         if strcmp(in, 'Params')
0044             ao = getDefaultPL();
0045             return
0046         elseif strcmp(in, 'Version')
0047             ao = VERSION;
0048             return
0049         end
0050     end
0051 end
0052
0053 if (nargout ~= 0)
0054
0055     hist_pl    = plist();
0056     fields     = fieldnames(ao);
0057     propArgin  = varargin;
0058
0059     if nargin >= 2 && isa(propArgin{1}, 'plist')
0060         pl      = propArgin{1};
0061         propArgin = {};
0062         for ii = 1:nparams(pl)
0063             %% The 'key' name of a plist/param is alway stored with upper case
0064             %% characters. To set a ao property with this key we have to lower the
0065             %% 'key' characters.

```

```

0066     propArgin{end+1} = lower(pl.params(ii).key);
0067     propArgin{end+1} =         pl.params(ii).val;
0068     end
0069 end
0070
0071
0072 while length(propArgin) >= 2
0073     prop = propArgin{1};
0074     val  = propArgin{2};
0075     propArgin = propArgin(3:end);
0076
0077     if ~ismember(prop, [fields; 't0'; 'fs'; 'xunits'; 'yunits'])
0078         error(['### ', prop, ' is not a valid analysis object property.']);
0079     else
0080
0081         %% For all analysis objects (ao can be a vector or a matrix)
0082         for jj = 1:numel(ao)
0083
0084             %% Also pass through some parameters to data object
0085             if strcmp(prop, 'fs')
0086                 d = ao(jj).data;
0087                 if isa(d, 'tsdata') || isa(d, 'fsdata')
0088                     d = set(d, 'fs', val);
0089                 else
0090                     error('### No ''fs'' property for this data type');
0091                 end
0092                 ao(jj).data = d;
0093
0094             elseif strcmp(prop, 't0')
0095                 d = ao(jj).data;
0096                 if isa(d, 'tsdata')
0097                     d = set(d, 't0', val);
0098                 else
0099                     error('### The data object is not from the class tsdata');
0100                 end
0101                 ao(jj).data = d;
0102
0103             elseif strcmp(prop, 'xunits')
0104                 d = ao(jj).data;
0105                 d = set(d, 'xunits', val);
0106                 ao(jj).data = d;
0107
0108             elseif strcmp(prop, 'yunits')
0109                 d = ao(jj).data;
0110                 d = set(d, 'yunits', val);
0111                 ao(jj).data = d;
0112
0113             else
0114                 ao(jj).(prop) = val;
0115             end

```

```

0116
0117     end % for
0118
0119 end
0120
0121 hist_pl = append(hist_pl, prop, val);
0122
0123 end
0124
0125 for jj = 1:numel(ao)
0126     h = history('set', VERSION, hist_pl);
0127     h = set(h, 'inhists', ao(jj).hist);
0128     h = set(h, 'invars', cellstr(inputname(1)));
0129
0130     ao(jj).hist = h;
0131 end
0132
0133 else
0134     if ischar(varargin{2})
0135         error('### please use: %s = set(%s, '%s'', '%s'');', ...
0136             inputname(1), ...
0137             inputname(1), ...
0138             varargin{1}, ...
0139             varargin{2});
0140     elseif isnumeric(varargin{2})
0141         error('### please use: %s = set(%s, '%s'', %d);', ...
0142             inputname(1), ...
0143             inputname(1), ...
0144             varargin{1}, ...
0145             varargin{2});
0146     else
0147         error('### please use: %s = set(%s, 'key'', 'value'');', ...
0148             inputname(1), ...
0149             inputname(1));
0150     end
0151
0152 end
0153
0154
0155 % Get default params
0156 function plo = getDefaultPL()
0157
0158 plo = plist();

```

[Home](#) > [classes](#) > [@ao](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a ao from a ao xml file.

SYNOPSIS

function ps = xmlparse(p, xmlfile)

DESCRIPTION

XMLPARSE creates a ao from a ao xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR overloads char() function for analysis objects.

This function is called by:

- [ao](#) AO analysis object class constructor.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a ao from a ao xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```

```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create ao from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'ao')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % 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')

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:

- **iplot** IPLOT provides an intelligent plotting tool for LTPDA.
- **plot_old** PLOT plots an array of analysis objects.

SUBFUNCTIONS

- **function** pl_default = getDefaultPL()

SOURCE CODE

```

0001 function ao_out = abs(varargin)
0002 % ABS overloads the Absolute value operator for Analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ABS overloads the Absolute value operator for Analysis objects.
0007 %
0008 % CALL: ao_out = abs(ao_in);
0009 %         ao_out = abs(ao_in, pl);
0010 %
0011 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0012 %                   ao_in  = ao_vector
0013 %                   ao_in  = ao_matrix
0014 %
0015 % PARAMETER LIST:  <key>          <value>          <description>
0016 %                   tsdata    fsdata    xydata
0017 %                   'xdata'    't'       'f'       'x'       compute the xdata
0018 %                   'ydata'    'x'       'xx'      'y'       compute the ydata
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = abs(ao, 'Params')
0024 %
0025 % HISTORY: 12-03-07 M Hewitson
0026 %           Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029 %
0030 VERSION  = '$Id: abs.m,v 1.11 2008/01/15 20:33:07 hewitson Exp $';
0031 CATEGORY = 'operator';
0032
0033 ao_out = [];
0034
0035 %% Check if this is a call for parameters

```

```

0036 if nargin == 2
0037     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0038         in = char(varargin{2});
0039         if strcmp(in, 'Params')
0040             ao_out = getDefaultPL();
0041             return
0042         elseif strcmp(in, 'Version')
0043             ao_out = VERSION;
0044             return
0045         elseif strcmp(in, 'Category')
0046             ao_out = CATEGORY;
0047             return
0048         end
0049     end
0050 end
0051
0052 %% Collect input ao's, plist's and ao variable names
0053 in_names = {};
0054 for ii = 1:nargin
0055     in_names{end+1} = inputname(ii);
0056 end
0057
0058 [aos, pls, invars] = collect_inputs(varargin, in_names);
0059
0060 if ~isempty (pls)
0061     pls = combine(pls);
0062 end
0063
0064 %% go through analysis objects
0065 for j=1:numel(aos)
0066     a = aos(j);
0067
0068     [h, abs_data] = single_operation(a.data, 'abs', pls);
0069
0070     %% Set the x, y-units to their old values (without abs())
0071     abs_data = set(abs_data, 'xunits', a.data.xunits);
0072     abs_data = set(abs_data, 'yunits', a.data.yunits);
0073     %% It is nicer with the absolute signs || than abs ()
0074     abs_data = set(abs_data, 'name', sprintf ('| %s |',a.data.name));
0075
0076     %% Add the history from the ao object to the history
0077     h = set(h, 'inhists', [a.hist]);
0078
0079     %% Set the var_name to the history
0080     h = set(h, 'invars', cellstr(invars{j}));
0081
0082     %% create a new analysis objects
0083     new_ao = ao (abs_data, h);
0084     new_ao = setnh(new_ao, 'name', sprintf ('| %s |',invars{j}) );
0085
0086     ao_out = [ao_out new_ao];

```

```
0087
0088 end
0089
0090 % Reshape the ouput to the same size of the input
0091 ao_out = reshape(ao_out, size(aos));
0092
0093
0094 %% Get default params
0095 function pl_default = getDefaultPL()
0096
0097     pl_default = plist([param('xdata', '')
0098                           param('ydata', '')] );
0099
0100 % END
```

Generated on Tue 22-Jan-2008 10:39:13 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')

HISTORY: 07-05-2007 Diepholz
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

```

0001 function ao_out = acos (varargin)
0002 % ACOS overloads the acos operator for Analysis objects. Inverse cosine result
in radians.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ACOS overloads the acos operator for Analysis objects.
0007 %               Inverse cosine result in radians.
0008 %               ACOS(ao) is the cosine of the elements of ao.data.
0009 %
0010 % CALL: ao_out = acos(ao_in);
0011 %       ao_out = acos(ao_in, pl);
0012 %       ao_out = acos(ao1, pl1, ao_vector, ao_matrix, pl2);
0013 %
0014 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0015 %                   ao_in  = ao_vector
0016 %                   ao_in  = ao_matrix
0017 %
0018 % PARAMETER LIST:  <key>          <value>          <description>
0019 %                   tsdata    fsdata    xydata
0020 %                   'xdata'    't'       'f'       'x'       compute the xdata
0021 %                   'ydata'    'x'       'xx'      'y'       compute the ydata
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 % >> pl = acos(ao, 'Params')
0027 %
0028 % HISTORY: 07-05-2007 Diepholz
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032
0033 VERSION = '$Id: acos.m,v 1.9 2007/11/26 14:48:13 ingo Exp $';
0034 ao_out  = [];

```

```

0035
0036 %% Check if this is a call for parameters
0037 if nargin == 2
0038     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0039         in = char(varargin{2});
0040         if strcmp(in, 'Params')
0041             ao_out = getDefaultPL();
0042             return
0043         elseif strcmp(in, 'Version')
0044             ao_out = VERSION;
0045             return
0046         end
0047     end
0048 end
0049
0050 %% Collect input ao's, plist's and ao variable names
0051 in_names = {};
0052 for ii = 1:nargin
0053     in_names{end+1} = inputname(ii);
0054 end
0055
0056 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0057
0058 if ~isempty(pl)
0059     pl = combine(pl);
0060 end
0061
0062 %% go through analysis objects
0063 for j=1:numel(ao_set)
0064     a = ao_set(j);
0065
0066     [h, acos_data] = single_operation(a.data, 'acos', pl);
0067
0068     %% Add the history from the ao object to the history
0069     h = set(h, 'inhists', [a.hist]);
0070
0071     %% Set the var_name to the history
0072     h = set(h, 'invars', cellstr(invars{j}));
0073
0074     %% create a new analysis objects
0075     new_ao = a;
0076     new_ao = ao (acos_data, h);
0077     new_ao = setnh(new_ao, 'name', sprintf('acos(%s)', invars{j})) );
0078
0079     ao_out = [ao_out new_ao];
0080
0081 end
0082
0083 % Reshape the output to the same size of the input
0084 ao_out = reshape(ao_out, size(ao_set));
0085

```

```
0086 %% Get default params
0087 function pl_default = getDefaultPL()
0088
0089     pl_default = plist([param('xdata', '')
0090                           param('ydata', '')] );
0091
0092 % END
```


[Home](#) > [classes](#) > [@ao](#) > aosplit.m

aosplit

PURPOSE

SPLIT an analysis object into the specified segments.

SYNOPSIS

function bo = split(varargin)

DESCRIPTION

SPLIT an analysis object into the specified segments.

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

Inputs:

a - input analysis object
pl - input parameter list (see below for parameters)

Outputs:

b - array of analysis objects

Parameters:

'times' - an array of start/stop times to split the input ao by
'frequencies' - an array of start/stop frequencies to split by
'samples' - an array of start/stop samples to split by
'N' - split into N contiguous pieces

If more than one splitting method is specified, the priority goes like the list above.

The time vector in the output A0 retains the original time values (i.e. it doesn't start from zero).

The splitting is done as $s \leq t < e$.

Arrays of start/stop values should be like: [s1 e1 s2 e2]

M Hewitson 02-03-07

\$Id: split.m,v 1.5 2007/05/14 08:21:12 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [ao](#) AO analysis object class constructor.
- [find](#) FIND particular samples that satisfy the input query and return a new AO.
- [set](#) SET set an analysis object property.
- [times](#) TIMES overloads .* operator for analysis objects.

This function is called by:

SOURCE CODE

```

0001 function bo = split(varargin)
0002
0003 % SPLIT an analysis object into the specified segments.
0004 %
0005 % >> b = split(a, pl)
0006 %
0007 % Inputs:
0008 %   a - input analysis object
0009 %   pl - input parameter list (see below for parameters)
0010 %
0011 % Outputs:
0012 %   b - array of analysis objects
0013 %
0014 % Parameters:
0015 %   'times'      - an array of start/stop times to split the input ao by
0016 %   'frequencies' - an array of start/stop frequencies to split by
0017 %   'samples'    - an array of start/stop samples to split by
0018 %   'N'          - split into N contiguous pieces
0019 %
0020 % If more than one splitting method is specified, the priority goes like
0021 % the list above.
0022 %
0023 % The time vector in the output AO retains the original time values
0024 % (i.e. it doesn't start from zero).
0025 %
0026 % The splitting is done as s<=t<e.
0027 %
0028 % Arrays of start/stop values should be like: [s1 e1 s2 e2 ....]
0029 %
0030 % M Hewitson 02-03-07
0031 %
0032 % $Id: split.m,v 1.5 2007/05/14 08:21:12 hewitson Exp $
0033 %
0034
0035
0036 % capture input variable names

```

```

0037 invars = {};
0038 for j=1:nargin
0039     if isa(varargin{j}, 'ao')
0040         invars = [invars cellstr(inputname(j))];
0041     end
0042 end
0043
0044 ALGONAME = mfilename;
0045 VERSION  = '$Id: split.m,v 1.5 2007/05/14 08:21:12 hewitson Exp $';
0046
0047 % Look at inputs
0048 if nargin < 1
0049     error('### Incorrect number of inputs.')
0050 end
0051 if nargin == 1
0052     as = varargin{1};
0053     pl = plist();
0054 end
0055 if nargin == 2
0056     as = varargin{1};
0057     pl = varargin{2};
0058 end
0059
0060 % Initialise output
0061 bo = [];
0062 % Unpack parameter list
0063 N      = find(pl, 'N');
0064 times  = find(pl, 'times');
0065 frequencies = find(pl, 'frequencies');
0066 samples    = find(pl, 'samples');
0067
0068 % look at input data
0069 d = as.data;
0070 dinfo = whos('d');
0071 % name for this object
0072 if isempty(invars{1})
0073     n1 = as.name;
0074 else
0075     n1 = invars{1};
0076 end
0077
0078 %---- Time series
0079 if isa(d, 'tsdata')
0080
0081     if ~isempty(times)
0082         disp('* splitting by time...');
0083
0084         % get data
0085         t = d.t;
0086         x = d.x;
0087

```

```

0088 % examine time list
0089 ntimes = length(times);
0090 if mod(ntimes, 2) ~= 0
0091     error('### please specify a start and stop for each interval.')
0092 end
0093
0094 % go over each interval now
0095 for i=1:2:ntimes
0096
0097     is = times(i);
0098     ie = times(i+1);
0099
0100     idx = find(t>=is & t <ie);
0101     xo = x(idx);
0102
0103 % Make output analysis object
0104 nameStr = sprintf('split(%s)', d.name);
0105
0106 % create new output data
0107 data = tsdata(t(idx), x(idx));
0108 data = set(data, 'name', nameStr);
0109 data = set(data, 'xunits', d.xunits);
0110 data = set(data, 'yunits', d.yunits);
0111 data = set(data, 't0', d.t0);
0112
0113 % create new output history
0114 h = history(ALGONAME, VERSION, plist(param('times', [is ie])), as.hist);
0115 h = set(h, 'invars', invars);
0116
0117 % make output analysis object
0118 b = ao(data, h);
0119
0120 % set name
0121 b = set(b, 'name', sprintf('split(%s)', n1));
0122
0123 % Add to output array
0124 bo = [bo b];
0125
0126
0127 end
0128
0129 elseif ~isempty(samples)
0130
0131     disp('* splitting by samples...');
0132     error('### I have not been written yet. Please code me up.');
```

```

0133
0134 elseif ~isempty(N)
0135     disp(sprintf('* splitting into %d chunks', N));
0136     error('### I have not been written yet. Please code me up.');
```

```

0137
```

```

0138     else
0139         error('### do not know how to split.')
0140     end
0141
0142 %----- Frequency Series
0143 elseif isa(d, 'fsdata')
0144
0145     if ~isempty(frequencies)
0146         disp('* splitting by frequency...');
0147         % get data
0148         f = d.f;
0149         xx = d.xx;
0150
0151         % examine time list
0152         nfreqs = length(frequencies);
0153         if mod(nfreqs, 2) ~= 0
0154             error('### please specify a start and stop for each interval.')
0155         end
0156
0157         % go over each interval now
0158         for i=1:2:nfreqs
0159
0160             is = frequencies(i);
0161             ie = frequencies(i+1);
0162             idx = find(f>=is & f <ie);
0163
0164             % Make output analysis object
0165             nameStr = sprintf('split(%s)', d.name);
0166
0167             % create new output data
0168             d = set(d, 'f', f(idx));
0169             d = set(d, 'xx', xx(idx));
0170
0171             % create new output history
0172             h = history(ALGONAME, VERSION, plist(param('frequencies', [is ie])), as.
hist);
0173             h = set(h, 'invars', invars);
0174
0175             % make output analysis object
0176             b = ao(d, h);
0177
0178             % set name
0179             b = set(b, 'name', sprintf('split(%s)', n1));
0180
0181             % Add to output array
0182             bo = [bo b];
0183
0184
0185         end
0186     elseif ~isempty(samples)
0187         disp('* splitting by samples...');

```

```
0188     error('### I have not been written yet. Please code me up.');
```

```
0189
```

```
0190     elseif ~isempty(N)
```

```
0191         disp(sprintf('* splitting into %d chunks', N));
```

```
0192         error('### I have not been written yet. Please code me up.');
```

```
0193
```

```
0194     else
```

```
0195         error('### do not know how to split.')
```

```
0196     end
```

```
0197
```

```
0198 else
```

```
0199     error('### unknown data type.');
```

```
0200 end
```

```
0201
```

```
0202
```

```
0203
```

```
0204 % END
```

```
0205
```

```
0206
```

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')

HISTORY: 07-05-2007 Diepholz
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

```

0001 function ao_out = asin (varargin)
0002 % ASIN overloads the asin operator for Analysis objects. Inverse sine result in
0003 % radians.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ASIN overloads the asin operator for Analysis objects.
0007 %               Inverse sine result in radians.
0008 %               ASIN(ao) is the arcsine of the elements of ao.data.
0009 %
0010 % CALL: ao_out = asin(ao_in);
0011 %       ao_out = asin(ao_in, pl);
0012 %       ao_out = asin(ao1, pl1, ao_vector, ao_matrix, pl2);
0013 %
0014 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0015 %                   ao_in  = ao_vector
0016 %                   ao_in  = ao_matrix
0017 %
0018 % PARAMETER LIST:  <key>          <value>          <description>
0019 %                   tsdata  fsdata  xydata
0020 %                   'xdata'  't'      'f'      'x'      compute the xdata
0021 %                   'ydata'  'x'      'xx'     'y'      compute the ydata
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 % >> pl = asin(ao, 'Params')
0027 %
0028 % HISTORY: 07-05-2007 Diepholz
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032 %
0033 VERSION = '$Id: asin.m,v 1.8 2007/11/26 14:48:13 ingo Exp $';
0034 ao_out  = [];

```



```

0035
0036 %% Check if this is a call for parameters
0037 if nargin == 2
0038     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0039         in = char(varargin{2});
0040         if strcmp(in, 'Params')
0041             ao_out = getDefaultPL();
0042             return
0043         elseif strcmp(in, 'Version')
0044             ao_out = VERSION;
0045             return
0046         end
0047     end
0048 end
0049
0050 %% Collect input ao's, plist's and ao variable names
0051 in_names = {};
0052 for ii = 1:nargin
0053     in_names{end+1} = inputname(ii);
0054 end
0055
0056 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0057
0058 if ~isempty(pl)
0059     pl = combine(pl);
0060 end
0061
0062 %% go through analysis objects
0063 for j=1:numel(ao_set)
0064     a = ao_set(j);
0065
0066     [h, asin_data] = single_operation(a.data, 'asin', pl);
0067
0068     %% Add the history from the ao object to the history
0069     h = set(h, 'inhists', [a.hist]);
0070
0071     %% Set the var_name to the history
0072     h = set(h, 'invars', cellstr(invars{j}));
0073
0074     %% create a new analysis objects
0075     new_ao = ao (asin_data, h);
0076     new_ao = setnh (new_ao, 'name', sprintf('asin(%s)', invars{j}));
0077
0078     ao_out = [ao_out new_ao];
0079
0080 end
0081
0082 % Reshape the output to the same size of the input
0083 ao_out = reshape(ao_out, size(ao_set));
0084
0085 %% Get default params
0086 function pl_default = getDefaultPL()

```

```
0087
0088     pl_default = plist([param('xdata', '')
0089                          param('ydata', '')]);
0090
0091 % END
```

Generated on Tue 22-Jan-2008 10:39:13 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')
```

HISTORY: 07-05-2007 Diepholz
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

```

0001 function ao_out = atan (varargin)
0002 % ATAN overloads the atan operator for Analysis objects. Inverse tangent result
in radians.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ATAN overloads the atan operator for Analysis objects.
0007 %               Inverse tangent result in radians.
0008 %               ATAN(ao) is the arctangent of the elements of ao.data.
0009 %
0010 % CALL: ao_out = atan(ao_in);
0011 %       ao_out = atan(ao_in, pl);
0012 %       ao_out = atan(ao1, pl1, ao_vector, ao_matrix, pl2);
0013 %
0014 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0015 %                   ao_in  = ao_vector
0016 %                   ao_in  = ao_matrix
0017 %
0018 % PARAMETER LIST:  <key>          <value>          <description>
0019 %                   tsdata  fsdata  xydata
0020 %                   'xdata'  't'      'f'      'x'      compute the xdata
0021 %                   'ydata'  'x'      'xx'     'y'      compute the ydata
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 % >> pl = atan(ao, 'Params')
0027 %
0028 % HISTORY: 07-05-2007 Diepholz
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032
0033 VERSION = '$Id: atan.m,v 1.8 2007/11/26 14:48:13 ingo Exp $';
0034 ao_out  = [];

```

```

0035
0036 %% Check if this is a call for parameters
0037 if nargin == 2
0038     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0039         in = char(varargin{2});
0040         if strcmp(in, 'Params')
0041             ao_out = getDefaultPL();
0042             return
0043         elseif strcmp(in, 'Version')
0044             ao_out = VERSION;
0045             return
0046         end
0047     end
0048 end
0049
0050 %% Collect input ao's, plist's and ao variable names
0051 in_names = {};
0052 for ii = 1:nargin
0053     in_names{end+1} = inputname(ii);
0054 end
0055
0056 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0057
0058 if ~isempty(pl)
0059     pl = combine(pl);
0060 end
0061
0062 %% go through analysis objects
0063 for j=1:numel(ao_set)
0064     a = ao_set(j);
0065
0066     [h, atan_data] = single_operation(a.data, 'atan', pl);
0067
0068     %% Add the history from the ao object to the history
0069     h = set(h, 'inhists', [a.hist]);
0070
0071     %% Set the var_name to the history
0072     h = set(h, 'invars', cellstr(invars{j}));
0073
0074     %% create a new analysis objects
0075     new_ao = ao (atan_data, h);
0076     new_ao = setnh (new_ao, 'name', sprintf('atan(%s)', invars{j}));
0077
0078     ao_out = [ao_out new_ao];
0079
0080 end
0081
0082 % Reshape the output to the same size of the input
0083 ao_out = reshape(ao_out, size(ao_set));
0084
0085 %% Get default params
0086 function pl_default = getDefaultPL()

```

```
0087
0088     pl_default = plist([param('xdata', '')
0089                          param('ydata', ')]);
0090
0091 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > complex.m

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')
```

VERSION: \$Id: complex.m,v 1.6 2007/11/09 17:53:59 ingo 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

```

0001 function ao_out = complex(a1, a2)
0002 % COMPLEX overloads the complex operator for Analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: COMPLEX overloads the complex operator for Analysis objects.
0007 %             A3 = COMPLEX(A1,A2) returns the complex result A + Bi, where A
and B
0008 %             are identically sized real arrays.
0009 %
0010 % CALL:       ao_out = complex(a1, a2);
0011 %
0012 % The following call returns a parameter list object that contains the
0013 % default parameter values:
0014 %
0015 % >> pl = imag(ao, 'Params')
0016 %
0017 % VERSION:    $Id: complex.m,v 1.6 2007/11/09 17:53:59 ingo Exp $
0018 %
0019 % HISTORY:    20-08-2007 Diepholz
0020 %             Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 ALGONAME = mfilename;
0025 VERSION  = '$Id: complex.m,v 1.6 2007/11/09 17:53:59 ingo Exp $';
0026
0027 % Check if this is a call for parameters
0028 if nargin == 2
0029     if isa(a1, 'ao') && ischar(a2)
0030         in = char(a2);
0031         if strcmp(in, 'Params')
0032             ao_out = plist();
0033             return
0034         elseif strcmp(in, 'Version')
0035             ao_out = VERSION;
0036             return
0037         end
0038     end
0039 end
0040
0041 % Check for the same data.
0042 if ~strcmp(class(a1.data), class(a2.data))
0043     error('### The data class must be the same. (%s <-> %s)', ...
0044         class(a1.data), class(a2.data));
0045 end

```



```

0046
0047 % Check for the same sampe rate
0048 fields = fieldnames(a1.data);
0049 if ismember('fs', fields)
0050     if a1.data.fs ~= a2.data.fs
0051         error('### The sample rate is not the same. Please resample one of the
A0's');
0052     end
0053 end
0054
0055 % Check the length of the A0's
0056 [x1,y1] = get_xy_values(a1.data);
0057 [x2,y2] = get_xy_values(a2.data);
0058 if length(x1) ~= length(x2) || length(y1) ~= length(y2)
0059     error('### The length of the data vectors must be the same.')
0060 end
0061
0062 complex_x = x1;
0063 complex_y = complex(y1, y2);
0064
0065 cmd = sprintf('data = %s(complex_x, complex_y);', class(a1.data));
0066 eval(cmd);
0067
0068 data = set(data, 'name', 'complex');
0069 data = set(data, 'xunits', a1.data.xunits);
0070 data = set(data, 'yunits', a1.data.yunits);
0071
0072 h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
0073 h = set(h, 'invars', {inputname(1), inputname(2)});
0074
0075 ao_out = ao(data, h);
0076 ao_out = setnh(ao_out, 'name', sprintf('complex(%s, %s)', inputname(1),
inputname(2)));
0077
0078
0079
0080
0081
0082
0083
0084
0085

```

[Home](#) > [classes](#) > [@ao](#) > conj.m

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 1.12 2007/11/26 14:48:13 ingo Exp \$

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

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

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

```

0001 function varargout = conj(varargin)
0002 % CONJ overloads the conjugate operator for Analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CONJ overloads the conjugate operator for Analysis objects.
0007 %
0008 % CALL:          a_conj = conj
0009 %
0010 % VERSION:       $Id: conj.m,v 1.12 2007/11/26 14:48:13 ingo Exp $
0011 %
0012 % The following call returns a parameter list object that contains the
0013 % default parameter values:
0014 %
0015 % >> pl = conj(ao, 'Params')
0016 %
0017 % HISTORY: 12-03-07 M Hewitson
0018 %          Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 % Check if this is a call for parameters
0023
0024 VERSION = '$Id: conj.m,v 1.12 2007/11/26 14:48:13 ingo Exp $';
0025 bs      = [];
0026
0027 if nargin == 2
0028     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0029         in = char(varargin{2});
0030         if strcmp(in, 'Params')
0031             varargout{1} = getDefaultPL();
0032             return
0033         elseif strcmp(in, 'Version')
0034             varargout{1} = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 %% Collect input ao's, plist's and ao variable names
0041 in_names = {};
0042 for ii = 1:nargin
0043     in_names{end+1} = inputname(ii);
0044 end
0045
0046 [as, ps, invars] = collect_inputs(varargin, in_names);

```

```

0047
0048 % check plist
0049 if isempty(ps)
0050     pl = getDefaultPL();
0051 else
0052     pl = combine(ps, getDefaultPL());
0053 end
0054
0055 %% go through analysis objects
0056 for j=1:numel(as)
0057     a = as(j);
0058
0059     [h, conj_data] = single_operation(a.data, 'conj', pl);
0060
0061     %% Set the x, y-units to their old values (without abs())
0062     conj_data = set(conj_data, 'xunits', a.data.xunits);
0063     conj_data = set(conj_data, 'yunits', a.data.yunits);
0064     conj_data = set(conj_data, 'name', sprintf('conj(%s)', a.data.name));
0065
0066     %% Add the history from the ao object to the history
0067     h = set(h, 'inhists', [a.hist]);
0068
0069     %% Set the var_name to the history
0070     h = set(h, 'invars', cellstr(invars{j}));
0071
0072     %% create a new analysis objects
0073     new_ao = ao (conj_data, h);
0074     new_ao = setnh(new_ao, 'name', sprintf('conj(%s)', invars{j}) );
0075
0076     % add to output
0077     bs = [bs new_ao];
0078
0079 end
0080
0081 % Reshape the output to the same size of the input
0082 varargout{1} = reshape(bs, size(as));
0083
0084 %% Get default params
0085 function plo = getDefaultPL()
0086 plo = plist();
0087
0088 % END

```

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')

HISTORY: 07-05-2007 Diepholz
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

```

0001 function ao_out = cos (varargin)
0002 % COS overloads the cos operator for Analysis objects. Cosine of argument in
0003 % radians.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: COS overloads the cos operator for Analysis objects.
0007 %               Cosine of argument in radians.
0008 %               COS(ao) is the cosine of the elements of ao.data.
0009 %
0010 % CALL: ao_out = cos(ao_in);
0011 %       ao_out = cos(ao_in, pl);
0012 %       ao_out = cos(ao1, pl1, ao_vector, ao_matrix, pl2);
0013 %
0014 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0015 %                   ao_in  = ao_vector
0016 %                   ao_in  = ao_matrix
0017 %
0018 % PARAMETER LIST:  <key>          <value>          <description>
0019 %                   tsdata  fsdata  xydata
0020 %                   'xdata'  't'      'f'      'x'      compute the xdata
0021 %                   'ydata'  'x'      'xx'     'y'      compute the ydata
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 % >> pl = cos(ao, 'Params')
0027 %
0028 % HISTORY: 07-05-2007 Diepholz
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032 %
0033 VERSION = '$Id: cos.m,v 1.8 2007/11/26 14:48:13 ingo Exp $';
0034 ao_out  = [];

```

```

0035
0036 %% Check if this is a call for parameters
0037 if nargin == 2
0038     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0039         in = char(varargin{2});
0040         if strcmp(in, 'Params')
0041             ao_out = getDefaultPL();
0042             return
0043         elseif strcmp(in, 'Version')
0044             ao_out = VERSION;
0045             return
0046         end
0047     end
0048 end
0049
0050 %% Collect input ao's, plist's and ao variable names
0051 in_names = {};
0052 for ii = 1:nargin
0053     in_names{end+1} = inputname(ii);
0054 end
0055
0056 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0057
0058 if ~isempty(pl)
0059     pl = combine(pl);
0060 end
0061
0062 %% go through analysis objects
0063 for j=1:numel(ao_set)
0064     a = ao_set(j);
0065
0066     [h, cos_data] = single_operation(a.data, 'cos', pl);
0067
0068     %% Add the history from the ao object to the history
0069     h = set(h, 'inhists', [a.hist]);
0070
0071     %% Set the var_name to the history
0072     h = set(h, 'invars', cellstr(invars{j}));
0073
0074     %% create a new analysis objects
0075     new_ao = a;
0076     new_ao = ao (cos_data, h);
0077     new_ao = setnh (new_ao, 'name', sprintf('cos(%s)', invars{j}) );
0078
0079     ao_out = [ao_out new_ao];
0080
0081 end
0082
0083 % Reshape the output to the same size of the input
0084 ao_out = reshape(ao_out, size(ao_set));
0085

```

```
0086 %% Get default params
0087 function pl_default = getDefaultPL()
0088
0089     pl_default = plist([param('xdata', '')
0090                           param('ydata', '')] );
0091
0092 % END
```


[Home](#) > [classes](#) > [@ao](#) > ctranpose.m

ctranpose

PURPOSE

CTRANSPOSE overloads the ' operator for Analysis Objects.

SYNOPSIS

function varargout = ctranpose(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: ctranpose.m,v 1.13 2007/12/21 10:41:32 hewitson Exp \$

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

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

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

```

0001 function varargout = ctranspose(varargin)
0002 % CTRANSPOSE overloads the ' operator for Analysis Objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CTRANSPOSE overloads the ' operator for Analysis Objects.
0007 %
0008 % CALL:          a = a1'      % only with data = cdata
0009 %
0010 % VERSION:       $Id: ctranspose.m,v 1.13 2007/12/21 10:41:32 hewitson Exp $
0011 %
0012 % The following call returns a parameter list object that contains the
0013 % default parameter values:
0014 %
0015 % >> pl = ctranspose(ao, 'Params')
0016 %
0017 % HISTORY: 08-05-07 A Monsky
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: ctranspose.m,v 1.13 2007/12/21 10:41:32 hewitson Exp $';
0023 bs      = [];
0024
0025 %% Check if this is a call for parameters
0026
0027 if nargin == 2
0028     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0029         in = char(varargin{2});
0030         if strcmp(in, 'Params')
0031             varargout{1} = getDefaultPL();
0032             return
0033         elseif strcmp(in, 'Version')
0034             varargout{1} = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 %% Collect input ao's, plist's and ao variable names
0041 in_names = {};
0042 for ii = 1:nargin
0043     in_names{end+1} = inputname(ii);
0044 end

```

```

0045
0046 [as, ps, invars] = collect_inputs(varargin, in_names);
0047
0048 % check plist
0049 if isempty(ps)
0050     pl = getDefaultPL();
0051 else
0052     pl = combine(ps, getDefaultPL());
0053 end
0054
0055 %% go through analysis objects
0056 for j=1:numel(as)
0057     a = as(j);
0058
0059     d = get(a, 'data');
0060     dinfo = whos('d');
0061     % Which data type do we have
0062     dtype = dinfo.class;
0063
0064     switch dtype
0065     case 'cdata'
0066 %         disp('* CTRANSPOSE of cdata object');
0067         % make a new cdata object
0068         [h, c] = single_operation(d, 'ctranspose', pl);
0069
0070         % make output analysis object
0071         h = set(h, 'inhists', [a.hist]);
0072
0073         %% Set the var_name to the history
0074         h = set(h, 'invars', cellstr(invars{j}));
0075
0076         b = ao(c, h);
0077         b = setnh(b, 'name', sprintf('ctranspose(%s)', char(invars{j})));
0078
0079     case {'tsdata', 'fsdata', 'xydata'}
0080         error('### this function works for cdata type A0 only')
0081     otherwise
0082         error('### unknown data type.')
0083
0084     end
0085
0086     % add to output
0087     bs = [bs b];
0088 end
0089
0090 % Reshape the ouput to the same size of the input
0091 bs = reshape(bs, size(as));
0092 varargout{1} = bs;
0093
0094 %% Get default params
0095 function plo = getDefaultPL()

```

```
0096  
0097 plo = plist();  
0098  
0099 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > decimate.m

decimate

PURPOSE

DECIMATE AOs containing time-series data.

SYNOPSIS

function bo = decimate(varargin)

DESCRIPTION

DECIMATE AOs containing time-series data.

%%%

DESCRIPTION: DECIMATE AOs containing time-series data.

CALL: b = decimate(a, 4) - decimate x4; offset is set to default of 0
 b = decimate(a, 2, 1) - decimate x2 with 1 sample offset
 b = decimate(a, pl) - use plist to get paramters
 b = decimate(a1, a2, pl) - decimate both a1 and a2; b is then a 2x1
 vector.

PARAMETERS: 'factor' - decimation factor
 'offset' - sample offset for decimation [default: 0]

VERSION: \$Id: decimate.m,v 1.9 2007/11/09 17:53:59 ingo Exp \$

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

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

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

```

0001 function bo = decimate(varargin)
0002 % DECIMATE AOs containing time-series data.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DECIMATE AOs containing time-series data.
0007 %
0008 % CALL:          b = decimate(a, 4)      - decimate x4; offset is set to default
of 0
0009 %
0010 %          b = decimate(a, 2, 1) - decimate x2 with 1 sample offset
0011 %          b = decimate(a, pl)   - use plist to get paramters
0012 %          b = decimate(a1, a2, pl) - decimate both a1 and a2; b is then a
2x1
0013 %
0014 %          vector.
0015 %
0016 %
0017 % PARAMETERS: 'factor' - decimation factor
0018 %              'offset' - sample offset for decimation [default: 0]
0019 %
0020 %
0021 % VERSION:      $Id: decimate.m,v 1.9 2007/11/09 17:53:59 ingo Exp $
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 %
0027 %
0028 %
0029 %
0030 %
0031 %
0032 %
0033 %
0034 %
0035 %
0036 %
0037 %
0038 %
0039 %
0040 %
0041 %
0042 %
0043 %
0044 %
0045 %
0046 %
0047 %
0048 %
0049 %
0050 %
0051 %
0052 %
0053 %
0054 %
0055 %
0056 %
0057 %
0058 %
0059 %
0060 %
0061 %
0062 %
0063 %
0064 %
0065 %
0066 %
0067 %
0068 %
0069 %
0070 %
0071 %
0072 %
0073 %
0074 %
0075 %
0076 %
0077 %
0078 %
0079 %
0080 %
0081 %
0082 %
0083 %
0084 %
0085 %
0086 %
0087 %
0088 %
0089 %
0090 %
0091 %
0092 %
0093 %
0094 %
0095 %
0096 %
0097 %
0098 %
0099 %
0100 %
0101 %
0102 %
0103 %
0104 %
0105 %
0106 %
0107 %
0108 %
0109 %
0110 %
0111 %
0112 %
0113 %
0114 %
0115 %
0116 %
0117 %
0118 %
0119 %
0120 %
0121 %
0122 %
0123 %
0124 %
0125 %
0126 %
0127 %
0128 %
0129 %
0130 %
0131 %
0132 %
0133 %
0134 %
0135 %
0136 %
0137 %
0138 %
0139 %
0140 %
0141 %
0142 %
0143 %
0144 %
0145 %
0146 %
0147 %
0148 %
0149 %
0150 %
0151 %
0152 %
0153 %
0154 %
0155 %
0156 %
0157 %
0158 %
0159 %
0160 %
0161 %
0162 %
0163 %
0164 %
0165 %
0166 %
0167 %
0168 %
0169 %
0170 %
0171 %
0172 %
0173 %
0174 %
0175 %
0176 %
0177 %
0178 %
0179 %
0180 %
0181 %
0182 %
0183 %
0184 %
0185 %
0186 %
0187 %
0188 %
0189 %
0190 %
0191 %
0192 %
0193 %
0194 %
0195 %
0196 %
0197 %
0198 %
0199 %
0200 %
0201 %
0202 %
0203 %
0204 %
0205 %
0206 %
0207 %
0208 %
0209 %
0210 %
0211 %
0212 %
0213 %
0214 %
0215 %
0216 %
0217 %
0218 %
0219 %
0220 %
0221 %
0222 %
0223 %
0224 %
0225 %
0226 %
0227 %
0228 %
0229 %
0230 %
0231 %
0232 %
0233 %
0234 %
0235 %
0236 %
0237 %
0238 %
0239 %
0240 %
0241 %
0242 %
0243 %
0244 %
0245 %
0246 %
0247 %
0248 %
0249 %
0250 %
0251 %
0252 %
0253 %
0254 %
0255 %
0256 %
0257 %
0258 %
0259 %
0260 %
0261 %
0262 %
0263 %
0264 %
0265 %
0266 %
0267 %
0268 %
0269 %
0270 %
0271 %
0272 %
0273 %
0274 %
0275 %
0276 %
0277 %
0278 %
0279 %
0280 %
0281 %
0282 %
0283 %
0284 %
0285 %
0286 %
0287 %
0288 %
0289 %
0290 %
0291 %
0292 %
0293 %
0294 %
0295 %
0296 %
0297 %
0298 %
0299 %
0300 %
0301 %
0302 %
0303 %
0304 %
0305 %
0306 %
0307 %
0308 %
0309 %
0310 %
0311 %
0312 %
0313 %
0314 %
0315 %
0316 %
0317 %
0318 %
0319 %
0320 %
0321 %
0322 %
0323 %
0324 %
0325 %
0326 %
0327 %
0328 %
0329 %
0330 %
0331 %
0332 %
0333 %
0334 %
0335 %
0336 %
0337 %
0338 %
0339 %
0340 %
0341 %
0342 %
0343 %
0344 %
0345 %
0346 %
0347 %
0348 %
0349 %
0350 %
0351 %
0352 %
0353 %
0354 %
0355 %
0356 %
0357 %
0358 %
0359 %
0360 %
0361 %
0362 %
0363 %
0364 %
0365 %
0366 %
0367 %
0368 %
0369 %
0370 %
0371 %
0372 %
0373 %
0374 %
0375 %
0376 %
0377 %
0378 %
0379 %
0380 %
0381 %
0382 %
0383 %
0384 %
0385 %
0386 %
0387 %
0388 %
0389 %
0390 %
0391 %
0392 %
0393 %
0394 %
0395 %
0396 %
0397 %
0398 %
0399 %
0400 %
0401 %
0402 %
0403 %
0404 %
0405 %
0406 %
0407 %
0408 %
0409 %
0410 %
0411 %
0412 %
0413 %
0414 %
0415 %
0416 %
0417 %
0418 %
0419 %
0420 %
0421 %
0422 %
0423 %
0424 %
0425 %
0426 %
0427 %
0428 %
0429 %
0430 %
0431 %
0432 %
0433 %
0434 %
0435 %
0436 %
0437 %
0438 %
0439 %
0440 %
0441 %
0442 %
0443 %
0444 %
0445 %
0446 %
0447 %
0448 %
0449 %
0450 %
0451 %
0452 %
0453 %
0454 %
0455 %
0456 %
0457 %
0458 %
0459 %
0460 %
0461 %
0462 %
0463 %
0464 %
0465 %
0466 %
0467 %
0468 %
0469 %
0470 %
0471 %
0472 %
0473 %
0474 %
0475 %
0476 %
0477 %
0478 %
0479 %
0480 %
0481 %
0482 %
0483 %
0484 %
0485 %
0486 %
0487 %
0488 %
0489 %
0490 %
0491 %
0492 %
0493 %
0494 %
0495 %
0496 %
0497 %
0498 %
0499 %
0500 %
0501 %
0502 %
0503 %
0504 %
0505 %
0506 %
0507 %
0508 %
0509 %
0510 %
0511 %
0512 %
0513 %
0514 %
0515 %
0516 %
0517 %
0518 %
0519 %
0520 %
0521 %
0522 %
0523 %
0524 %
0525 %
0526 %
0527 %
0528 %
0529 %
0530 %
0531 %
0532 %
0533 %
0534 %
0535 %
0536 %
0537 %
0538 %
0539 %
0540 %
0541 %
0542 %
0543 %
0544 %
0545 %
0546 %
0547 %
0548 %
0549 %
0550 %
0551 %
0552 %
0553 %
0554 %
0555 %
0556 %
0557 %
0558 %
0559 %
0560 %
0561 %
0562 %
0563 %
0564 %
0565 %
0566 %
0567 %
0568 %
0569 %
0570 %
0571 %
0572 %
0573 %
0574 %
0575 %
0576 %
0577 %
0578 %
0579 %
0580 %
0581 %
0582 %
0583 %
0584 %
0585 %
0586 %
0587 %
0588 %
0589 %
0590 %
0591 %
0592 %
0593 %
0594 %
0595 %
0596 %
0597 %
0598 %
0599 %
0600 %
0601 %
0602 %
0603 %
0604 %
0605 %
0606 %
0607 %
0608 %
0609 %
0610 %
0611 %
0612 %
0613 %
0614 %
0615 %
0616 %
0617 %
0618 %
0619 %
0620 %
0621 %
0622 %
0623 %
0624 %
0625 %
0626 %
0627 %
0628 %
0629 %
0630 %
0631 %
0632 %
0633 %
0634 %
0635 %
0636 %
0637 %
0638 %
0639 %
0640 %
0641 %
0642 %
0643 %
0644 %
0645 %
0646 %
0647 %
0648 %
0649 %
0650 %
0651 %
0652 %
0653 %
0654 %
0655 %
0656 %
0657 %
0658 %
0659 %
0660 %
0661 %
0662 %
0663 %
0664 %
0665 %
0666 %
0667 %
0668 %
0669 %
0670 %
0671 %
0672 %
0673 %
0674 %
0675 %
0676 %
0677 %
0678 %
0679 %
0680 %
0681 %
0682 %
0683 %
0684 %
0685 %
0686 %
0687 %
0688 %
0689 %
0690 %
0691 %
0692 %
0693 %
0694 %
0695 %
0696 %
0697 %
0698 %
0699 %
0700 %
0701 %
0702 %
0703 %
0704 %
0705 %
0706 %
0707 %
0708 %
0709 %
0710 %
0711 %
0712 %
0713 %
0714 %
0715 %
0716 %
0717 %
0718 %
0719 %
0720 %
0721 %
0722 %
0723 %
0724 %
0725 %
0726 %
0727 %
0728 %
0729 %
0730 %
0731 %
0732 %
0733 %
0734 %
0735 %
0736 %
0737 %
0738 %
0739 %
0740 %
0741 %
0742 %
0743 %
0744 %
0745 %
0746 %
0747 %
0748 %
0749 %
0750 %
0751 %
0752 %
0753 %
0754 %
0755 %
0756 %
0757 %
0758 %
0759 %
0760 %
0761 %
0762 %
0763 %
0764 %
0765 %
0766 %
0767 %
0768 %
0769 %
0770 %
0771 %
0772 %
0773 %
0774 %
0775 %
0776 %
0777 %
0778 %
0779 %
0780 %
0781 %
0782 %
0783 %
0784 %
0785 %
0786 %
0787 %
0788 %
0789 %
0790 %
0791 %
0792 %
0793 %
0794 %
0795 %
0796 %
0797 %
0798 %
0799 %
0800 %
0801 %
0802 %
0803 %
0804 %
0805 %
0806 %
0807 %
0808 %
0809 %
0810 %
0811 %
0812 %
0813 %
0814 %
0815 %
0816 %
0817 %
0818 %
0819 %
0820 %
0821 %
0822 %
0823 %
0824 %
0825 %
0826 %
0827 %
0828 %
0829 %
0830 %
0831 %
0832 %
0833 %
0834 %
0835 %
0836 %
0837 %
0838 %
0839 %
0840 %
0841 %
0842 %
0843 %
0844 %
0845 %
0846 %
0847 %
0848 %
0849 %
0850 %
0851 %
0852 %
0853 %
0854 %
0855 %
0856 %
0857 %
0858 %
0859 %
0860 %
0861 %
0862 %
0863 %
0864 %
0865 %
0866 %
0867 %
0868 %
0869 %
0870 %
0871 %
0872 %
0873 %
0874 %
0875 %
0876 %
0877 %
0878 %
0879 %
0880 %
0881 %
0882 %
0883 %
0884 %
0885 %
0886 %
0887 %
0888 %
0889 %
0890 %
0891 %
0892 %
0893 %
0894 %
0895 %
0896 %
0897 %
0898 %
0899 %
0900 %
0901 %
0902 %
0903 %
0904 %
0905 %
0906 %
0907 %
0908 %
0909 %
0910 %
0911 %
0912 %
0913 %
0914 %
0915 %
0916 %
0917 %
0918 %
0919 %
0920 %
0921 %
0922 %
0923 %
0924 %
0925 %
0926 %
0927 %
0928 %
0929 %
0930 %
0931 %
0932 %
0933 %
0934 %
0935 %
0936 %
0937 %
0938 %
0939 %
0940 %
0941 %
0942 %
0943 %
0944 %
0945 %
0946 %
0947 %
0948 %
0949 %
0950 %
0951 %
0952 %
0953 %
0954 %
0955 %
0956 %
0957 %
0958 %
0959 %
0960 %
0961 %
0962 %
0963 %
0964 %
0965 %
0966 %
0967 %
0968 %
0969 %
0970 %
0971 %
0972 %
0973 %
0974 %
0975 %
0976 %
0977 %
0978 %
0979 %
0980 %
0981 %
0982 %
0983 %
0984 %
0985 %
0986 %
0987 %
0988 %
0989 %
0990 %
0991 %
0992 %
0993 %
0994 %
0995 %
0996 %
0997 %
0998 %
0999 %
1000 %

```

```

0036     if strcmp(in, 'Params')
0037         bo = getDefaultPL();
0038         return
0039     elseif strcmp(in, 'Version')
0040         bo = VERSION;
0041         return
0042     end
0043 end
0044 end
0045
0046 invars = {};
0047 as      = [];
0048 ps      = [];
0049 aos_in = 0;
0050
0051 %% collect input ao's, plist's and ao variable names
0052
0053 for j=1:nargin
0054
0055     if isa(varargin{j}, 'ao')
0056         as = [as varargin{j}];
0057         aos_in = aos_in + 1;
0058
0059         ao_name = inputname(j);
0060         if isempty(ao_name)
0061             ao_name = 'no_ao_name';
0062         end
0063
0064         % Memorise the variable name of the corresponding analysis object.
0065         % If the ao is an array or vector add the index to the variable name
0066         if numel(varargin{j}) == 1
0067             invars{end+1} = ao_name;
0068         else
0069             for ii=1:numel(varargin{j})
0070                 [I,J] = ind2sub(size(varargin{j}),ii);
0071                 invars{end+1} = sprintf('%s(%d,%d)', ao_name, I, J);
0072             end
0073         end
0074     end
0075     if isa(varargin{j}, 'plist')
0076         ps = [ps varargin{j}];
0077     end
0078 end
0079
0080 Na = numel(as);
0081 if isempty(as)
0082     error('### Please input at least one A0.');
```

```

0088 else
0089     pl = plist();
0090 end
0091
0092 %% Get parameters from plist
0093 offset = find(pl, 'offset');
0094 factor = find(pl, 'factor');
0095
0096 if rem(factor, floor(factor)) ~= 0
0097     warning('!!! Downsample factor should be an integer. Rounding. !!!');
0098     factor = round(factor);
0099 end
0100
0101 if nargin-aos_in == 1
0102     if isnumeric(varargin{end})
0103         factor = varargin{end};
0104     end
0105 end
0106 if nargin-aos_in == 2
0107     if isnumeric(varargin{end-1})
0108         factor = varargin{end-1};
0109     end
0110     if isnumeric(varargin{end})
0111         offset = varargin{end};
0112     end
0113 end
0114
0115 if isempty(factor)
0116     error('### Please specify a decimation factor either directly or in a
plist. ');
0117 end
0118 if isempty(offset)
0119     warning('!!! No offset specified; using default of 0 samples !!!');
0120     offset = 0;
0121 end
0122
0123 %% Loop over input AOs
0124 bo = [];
0125 for j=1:Na
0126
0127     a = as(j);
0128     d = a.data;
0129     x = d.x;
0130     y = d.y;
0131     fs = d.fs;
0132     if isa(d, 'tsdata')
0133         ss = 1+offset;
0134         samples = ss:factor:len(a);
0135         d = set(d, 'x', x(samples));
0136         d = set(d, 'y', y(samples));
0137         d = set(d, 'fs', fs/factor);

```



```

0138     else
0139         error('### I can only decimate time-series AOs.');
```

0140 end

0141

0142 %----- Make output AO

0143

0144 % create new output history

0145 plo = plist([param('factor', factor) param('offset', offset)]);

0146 h = history(ALGONAME, VERSION, plo, a.hist);

0147 h = set(h, 'invars', invars);

0148

0149 % make output analysis object

0150 b = ao(d, h);

0151

0152 % set name

0153 b = setnh(b, 'name', sprintf('decimate(%s)', invars{j}));

0154

0155 % Add to output array

0156 bo = [bo b];

0157

0158 end

0159

0160 % Reshape the output to the same size of the input

0161 bo = reshape(bo, size(as));

0162

0163 %% Get default params

0164 function pl_default = getDefaultPL()

0165

0166 pl_default = plist([param('factor', '')

0167 param('offset', 0)]);

0168

0169

0170 % END

[Home](#) > [classes](#) > [@ao](#) > det.m

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 1.15 2007/12/24 19:48:38 hewitson Exp \$

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

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

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

```

0001 function varargout = det(varargin)
0002 % DET overloads the determinant function for Analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DET overloads the determinant function for Analysis objects.
0007 %
0008 % CALL:          a = det(a1)    % only with data = cdata
0009 %
0010 % VERSION:       $Id: det.m,v 1.15 2007/12/24 19:48:38 hewitson Exp $
0011 %
0012 % The following call returns a parameter list object that contains the
0013 % default parameter values:
0014 %
0015 % >> pl = det(ao, 'Params')
0016 %
0017 % HISTORY: 08-05-07 A Monsky
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: det.m,v 1.15 2007/12/24 19:48:38 hewitson Exp $';
0023 bs      = [];
0024
0025 %% Check if this is a call for parameters
0026 if nargin == 2
0027     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0028         in = char(varargin{2});
0029         if strcmp(in, 'Params')
0030             varargout{1} = getDefaultPL();
0031             return
0032         elseif strcmp(in, 'Version')
0033             varargout{1} = VERSION;
0034             return
0035         end
0036     end
0037 end
0038
0039 %% Collect input ao's, plist's and ao variable names
0040 in_names = {};
0041 for ii = 1:nargin
0042     in_names{end+1} = inputname(ii);
0043 end
0044

```

```

0045 [as, ps, invars] = collect_inputs(varargin, in_names);
0046
0047 % check plist
0048 if isempty(ps)
0049     pl = getDefaultPL();
0050 else
0051     pl = combine(ps, getDefaultPL());
0052 end
0053
0054 %% go through analysis objects
0055 for j=1:numel(as)
0056     a = as(j);
0057
0058     d = get(a, 'data');
0059     dinfo = whos('d');
0060     % Which data type do we have
0061     dtype = dinfo.class;
0062
0063     switch dtype
0064     case 'cdata'
0065         % make a new cdata object
0066
0067         if size(d.y,1) == size(d.y,2)
0068             [h, c] = single_operation(d, 'det', pl);
0069         else
0070             error('### The value (y) of data must be a square matrix.')
0071         end
0072
0073         % make output analysis object
0074         h = set(h, 'inhists', [a.hist]);
0075
0076         %% Set the var_name to the history
0077         h = set(h, 'invars', cellstr(invars{j}));
0078
0079         b = ao(c, h);
0080         b = setnh(b, 'name', sprintf('det(%s)', char(invars{j})));
0081     case {'tsdata','fsdata','xydata'}
0082         error('### this function works for cdata type A0 only')
0083     otherwise
0084         error('### unknown data type.')
0085     end
0086
0087     % add to output
0088     bs = [bs b];
0089 end
0090
0091 bs = reshape(bs, size(as));
0092 varargout{1} = bs;
0093
0094 %% Get default params
0095 function plo = getDefaultPL()

```

```
0096  
0097 plo = plist();  
0098  
0099  
0100 % END
```

Generated on Tue 22-Jan-2008 10:39:13 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.

[illegible]

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 positive number.
e.g. `pl = plist('dim', 1)`; This parameter list computes the first 'sub' diagonal.

```
VERSION: $Id: diag.m,v 1.17 2008/01/10 20:00:29 hewitson Exp $
```

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

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

HISTORY: 08-05-07 A Monsky
Creation

[illegible]

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

```

0001 function varargout = diag(varargin)
0002 % DIAG overloads the diagonal operator for Analysis Objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DIAG overloads the diagonal operator for Analysis Objects.
0007 %
0008 % CALL:          b = diag (a)      % only with data = cdata
0009 %               b = diag (a, pl)
0010 %
0011 % INPUTS:        a - input analysis object
0012 %               pl - parameter list
0013 %
0014 % OUTPUTS:       b - output analysis object
0015 %
0016 % Parameters:    'dim' - The value of the dimension must be zero or a positiv
0017 %               number.
0018 %               e.g. pl = plist('dim', 1); This parameter list computes
0019 %               the first 'sub' diagonal.
0020 %
0021 % VERSION:       $Id: diag.m,v 1.17 2008/01/10 20:00:29 hewitson Exp $
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 % >> pl = diag(ao, 'Params')
0027 %
0028 % HISTORY: 08-05-07 A Monsky
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032
0033 VERSION = '$Id: diag.m,v 1.17 2008/01/10 20:00:29 hewitson Exp $';
0034

```

```

0035 %% Check if this is a call for parameters
0036
0037 if nargin == 2
0038     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0039         in = char(varargin{2});
0040         if strcmp(in, 'Params')
0041             varargout{1} = getDefaultPL();
0042             return
0043         elseif strcmp(in, 'Version')
0044             varargout{1} = VERSION;
0045             return
0046         end
0047     end
0048 end
0049
0050 %% capture input variable names
0051 invars = {};
0052
0053 as = [];
0054 bs = [];
0055 pl = [];
0056
0057 for j=1:nargin
0058     if isa(varargin{j}, 'ao')
0059         as = [as varargin{j}];
0060
0061         % Memorise the variable name of the corresponding analysis object.
0062         % If the ao is an array or vector add the index to the variable name
0063         if numel(varargin{j}) == 1
0064             invars{end+1} = inputname(j);
0065         else
0066             for ii=1:numel(varargin{j})
0067                 [I,J] = ind2sub(size(varargin{j}),ii);
0068                 invars{end+1} = sprintf('%s(%d,%d)', inputname(j), I, J);
0069             end
0070         end
0071     end
0072     if isa(varargin{j}, 'plist')
0073         pl = [pl varargin{j}];
0074     end
0075 end
0076
0077 if ~isempty(pl)
0078     pl = combine(pl);
0079 end
0080
0081
0082 %% go through analysis objects
0083 for j=1:numel(as)
0084     a = as(j);
0085
0086     d = get(a, 'data');

```



```

0087     dinfo = whos('d');
0088
0089     % Which data type do we have
0090     dtype = dinfo.class;
0091     switch dtype
0092
0093         case 'cdata'
0094             %disp('* DIAGONAL of cdata object');
0095             % make a new cdata object
0096             [h, c] = single_operation(d, 'diag',pl);
0097
0098             % make output analysis object
0099             h = set(h, 'inhists', [a.hist]);
0100
0101             h = set(h, 'invars', cellstr(invars{j}));
0102
0103             b = ao(c, h);
0104
0105             b = setnh(b, 'name', sprintf('diag(%s)', invars{j}));
0106
0107         case {'tsdata','fsdata','xydata'}
0108             error('### this function works for cdata type A0 only')
0109         otherwise
0110             error('### unknown data type.')
0111     end
0112
0113     % add to output
0114     bs = [bs b];
0115 end
0116
0117 % Reshape the ouput to the same size of the input
0118 bs = reshape(bs, size(as));
0119 varargout{1} = bs;
0120
0121 %% Get default params
0122 function plo = getDefaultPL()
0123
0124 plo = plist();
0125 plo = append(plo, 'option', 0);
0126
0127 % END

```

[Home](#) > [classes](#) > [@ao](#) > display.m

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 1.14 2008/01/08 16:39:11 ingo 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:

SOURCE CODE

```
0001 function txt = display(ao, varargin)
0002 % DISPLAY implement terminal display for analysis object.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY implement terminal display for analysis object.
0007 %
0008 % CALL:          txt = display(ao)
0009 %                  ao                  % without a semicolon
0010 %
0011 % VERSION:       $Id: display.m,v 1.14 2008/01/08 16:39:11 ingo Exp $
0012 %
0013 % HISTORY:       31-01-2007 M Hewitson
0014 %                  Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: display.m,v 1.14 2008/01/08 16:39:11 ingo Exp $';
0019
0020 %% Check if this is a call for parameters
0021 if nargin == 2
0022     if isa(ao, 'ao') && ischar(varargin{1})
0023         in = char(varargin{1});
0024         if strcmp(in, 'Params')
0025             txt = plist();
0026             return
0027         elseif strcmp(in, 'Version')
0028             txt = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 txt = {};
0035
0036 for i=1:numel(ao)
0037     a = ao(i);
0038     data    = get(a, 'data');
0039     dinfo   = whos('data');
0040     hist    = get(a, 'hist');
0041     hinfo   = whos('hist');
0042
0043     if nargin ~= length(ao)
0044         banner_start = sprintf('----- ao %02d -----', i);
0045     else
0046         banner_start = sprintf('----- ao: %s -----', inputname(i));
0047     end
0048
0049     txt{end+1} = banner_start;
0050     txt{end+1} = ' ';
0051     txt{end+1} = sprintf('          name: %s', get(a, 'name'));
0052     txt{end+1} = sprintf(' provenance: %s', char(get(a, 'provenance')));
0053     txt{end+1} = sprintf('description: %s', char(get(a, 'description')));
0054     if isa(data, 'cell')

```

```
0055     txt{end+1} = sprintf('          data:  None');
0056     else
0057         txt{end+1} = sprintf('          data:  %s / %s', dinfo.class, get(data,
'name'));
0058     end
0059     if isa(hist, 'cell')
0060         txt{end+1} = sprintf('          hist:  None');
0061     else
0062         txt{end+1} = sprintf('          hist:  %s / %s / %s', hinfo.class, get(hist,
'name'), get(hist, 'version'));
0063     end
0064     txt{end+1} = sprintf('  mfilename:  %s', a.mfilename);
0065
0066     banner_end(1:length(banner_start)) = '-';
0067     txt{end+1} = banner_end;
0068
0069     txt{end+1} = ' ';
0070     txt{end+1} = ' ';
0071 end
0072
0073 if nargout == 0
0074     for ii=1:length(txt)
0075         disp(txt{ii});
0076     end
0077 end
0078
0079 % END
```

[Home](#) > [classes](#) > [@ao](#) > eig.m

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 = svd (a,pl) % only with data = cdata
 [v, d] = svd (a,pl)
 [v, d] = svd (a)

INPUTS: pl - a parameter list
 a - input analysis object

OUTPUTS: like matlab fct

Parameters: 'option' - a string or value that can be submitted
 i.e. 'nobalance' to disable balancing options are the
 same as for the matlab function.

VERSION: \$Id: eig.m,v 1.17 2008/01/10 21:25:04 hewitson Exp \$

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

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

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

```

0001 function varargout = eig(varargin)
0002 % EIG overloads the determinant function for Analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EIG overloads the determinant function for Analysis objects.
0007 %
0008 % CALL:          v      = svd (a,pl)      % only with data = cdata
0009 %               [v, d] = svd (a,pl)
0010 %               [v, d] = svd (a)
0011 %
0012 % INPUTS:        pl    - a parameter list
0013 %               a      - input analysis object
0014 %
0015 % OUTPUTS:       like matlab fct
0016 %
0017 % Parameters:    'option' - a string or value that can be submitted
0018 %                                   i.e. 'nobalance' to disable balancing options are the
0019 %                                   same as for the matlab function.
0020 %
0021 % VERSION:       $Id: eig.m,v 1.17 2008/01/10 21:25:04 hewitson Exp $
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 % >> pl = eig(ao, 'Params')
0027 %
0028 % HISTORY: 08-05-07 A Monsky
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032
0033 VERSION = '$Id: eig.m,v 1.17 2008/01/10 21:25:04 hewitson Exp $';
0034 vdata   = [];

```

```

0035 ddata    = [];
0036 V        = [];
0037 D        = [];
0038
0039 %% Check if this is a call for parameters
0040
0041 if nargin == 2
0042     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0043         in = char(varargin{2});
0044         if strcmp(in, 'Params')
0045             varargout{1} = getDefaultPL();
0046             return
0047         elseif strcmp(in, 'Version')
0048             varargout{1} = VERSION;
0049             return
0050         end
0051     end
0052 end
0053
0054 %% Collect input ao's, plist's and ao variable names
0055 in_names = {};
0056 for ii = 1:nargin
0057     in_names{end+1} = inputname(ii);
0058 end
0059
0060 [as, pl, invars] = collect_inputs(varargin, in_names);
0061
0062 if ~isempty(pl)
0063     pl = combine(pl);
0064 end
0065
0066 %% Loop over analysis objects
0067 for j=1:numel(as)
0068     a = as(j);
0069
0070     d = get(a, 'data');
0071     dinfo = whos('d');
0072     % Which data type do we have
0073     dtype = dinfo.class;
0074
0075     switch dtype
0076     case 'cdata'
0077         if size(d.y,1) ~= size(d.y,2)
0078             error('### The value (y) of data must be a square matrix.')
0079         else
0080             %disp('* Eigenvalues of cdata object');
0081             if nargout <= 1
0082                 [h, vdata] = single_operation(d, 'eig',pl);
0083             elseif nargout == 2
0084                 [h, vdata, ddata] = single_operation(d, 'eig',pl);
0085             end
0086             %create analysis object(s)

```

```

0087         h = set(h, 'inhists', [a.hist]);
0088
0089         %% Set the var_name to the history
0090         h = set(h, 'invars', cellstr(invars{j}));
0091
0092         v = ao(vdata, h);
0093         v = setnh(v, 'name', sprintf('v_eig(%s)', char(invars{j})));
0094         V = [V v];
0095         if ~isempty(ddata)
0096             d = ao(ddata, h);
0097             d = setnh(d, 'name', sprintf('d_eig(%s)', char(invars{j})));
0098             D = [D d];
0099         end
0100     end
0101     case {'tsdata', 'fsdata', 'xydata'}
0102         error('### this function works for cdata type A0 only')
0103     otherwise
0104         error('### unknown data type.')
0105     end
0106 end
0107
0108 % Reshape the ouput to the same size of the input
0109 V = reshape(V, size(as));
0110 varargout{1} = V;
0111
0112 if nargin > 1
0113     % Reshape the ouput to the same size of the input
0114     D = reshape(D, size(as));
0115     varargout{2} = D;
0116     if nargin > 2
0117         error('### wrong number of outputs')
0118     end
0119 end
0120
0121 %% Get default params
0122 function plo = getDefaultPL()
0123
0124 plo = plist();
0125 plo = append(plo, 'option', '');
0126
0127 % END

```


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.

%%%

DESCRIPTION: EXP overloads the exp operator for Analysis objects. Exponential.
EXP(ao) is the exponential of the elements of ao.data
e to the ao.data.

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

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

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 = exp(ao, 'Params')

HISTORY: 23-05-2007 Diepholz
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

```

0001 function ao_out = exp (varargin)
0002 % EXP overloads the exp operator for Analysis objects. Exponential.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EXP overloads the exp operator for Analysis objects. Exponential.
0007 %               EXP(ao) is the exponential of the elements of ao.data
0008 %               e to the ao.data.
0009 %
0010 % CALL: ao_out = exp(ao_in);
0011 %       ao_out = exp(ao_in, pl);
0012 %       ao_out = exp(ao1, pl1, ao_vector, ao_matrix, pl2);
0013 %
0014 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0015 %                   ao_in  = ao_vector
0016 %                   ao_in  = ao_matrix
0017 %
0018 % PARAMETER LIST:  <key>          <value>          <description>
0019 %                   tsdata  fsdata  xydata
0020 %                   'xdata'  't'      'f'      'x'      compute the xdata
0021 %                   'ydata'  'x'      'xx'     'y'      compute the ydata
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 % >> pl = exp(ao, 'Params')
0027 %
0028 % HISTORY: 23-05-2007 Diepholz
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032 %
0033 VERSION = '$Id: exp.m,v 1.12 2007/11/26 14:48:13 ingo Exp $';
0034 ao_out  = [];
0035
0036 %% Check if this is a call for parameters
0037 if nargin == 2

```

```

0038     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0039         in = char(varargin{2});
0040         if strcmp(in, 'Params')
0041             ao_out = getDefaultPL();
0042             return
0043         elseif strcmp(in, 'Version')
0044             ao_out = VERSION;
0045             return
0046         end
0047     end
0048 end
0049
0050 %% Collect input ao's, plist's and ao variable names
0051 in_names = {};
0052 for ii = 1:nargin
0053     in_names{end+1} = inputname(ii);
0054 end
0055
0056 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0057
0058 if ~isempty (pl)
0059     pl = combine(pl);
0060 end
0061
0062 %% go through analysis objects
0063 for j=1:numel(ao_set)
0064     a = ao_set(j);
0065
0066     [h, exp_data] = single_operation(a.data, 'exp', pl);
0067
0068     %% Add the history from the ao object to the history
0069     h = set(h, 'inhists', [a.hist]);
0070
0071     %% Set the var_name to the history
0072     h = set(h, 'invars', cellstr(invars{j}));
0073
0074     %% create a new analysis objects
0075     new_ao = a;
0076     new_ao = ao (exp_data, h);
0077     new_ao = setnh (new_ao, 'name', sprintf('exp(%s)', invars{j}) );
0078
0079     ao_out = [ao_out new_ao];
0080
0081 end
0082
0083 % Reshape the ouput to the same size of the input
0084 ao_out = reshape(ao_out, size(ao_set));
0085
0086 %% Get default params
0087 function pl_default = getDefaultPL()
0088
0089     pl_default = plist([param('xdata', '')

```

```
0090          param('ydata', ''));  
0091  
0092  
0093 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > [fft.m](#)

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 1.10 2008/01/20 12:53:29 hewitson Exp \$

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

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

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

```

0001 function varargout = fft(varargin)
0002 % FFT overloads the fft operator for Analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: FFT overloads the fft operator for Analysis objects.
0007 %
0008 % CALL:          b = fft(a)
0009 %
0010 % VERSION:       $Id: fft.m,v 1.10 2008/01/20 12:53:29 hewitson Exp $
0011 %
0012 % The following call returns a parameter list object that contains the
0013 % default parameter values:
0014 %
0015 % >> pl = fft(ao, 'Params')
0016 %
0017 % HISTORY: 12-03-07 M Hewitson
0018 %          Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 ALGONAME = mfilename;
0023 VERSION  = '$Id: fft.m,v 1.10 2008/01/20 12:53:29 hewitson Exp $';
0024 bs       = [];
0025
0026 % Check if this is a call for parameters
0027 if nargin == 2
0028     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0029         in = char(varargin{2});
0030         if strcmp(in, 'Params')
0031             varargout{1} = getDefaultPL();
0032             return
0033         elseif strcmp(in, 'Version')
0034             varargout{1} = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 % Collect input ao's, plist's and ao variable names
0041 in_names = {};
0042 for ii = 1:nargin

```

```

0043     in_names{end+1} = inputname(ii);
0044 end
0045
0046 [as, ps, invars] = collect_inputs(varargin, in_names);
0047
0048
0049 % Check input analysis object
0050 for j=1:numel(as)
0051     a      = as(j);
0052     d      = get(a, 'data');
0053     dinfo = whos('d');
0054
0055     % Which data type do we have
0056     dtype = dinfo.class;
0057     switch dtype
0058     case 'tsdata'
0059
0060         % make a new fsdata object
0061         nfft = length(d.y);
0062         ft   = fft(d.y);
0063         ft   = ft(1:nfft/2+1);
0064         f     = linspace(0, d.fs/2, length(ft));
0065         fsd = fsdata(f, ft, d.fs);
0066         fsd = set(fsd, 'name', sprintf('fft(%s)', d.name));
0067         fsd = set(fsd, 'xunits', 'Hz');
0068         fsd = set(fsd, 'yunits', '');
0069
0070         % make a new history object
0071         h = history(ALGONAME, VERSION, plist(), [a.hist]);
0072         h = set(h, 'invars', cellstr(invars{j}));
0073
0074         % make output analysis object
0075         b = ao(fsd, h);
0076         b = setnh(b, 'name', sprintf('fft(%s)', invars{j}));
0077
0078     case {'fsdata', 'cdata', 'xydata'}
0079         error('### I don''t work for frequency-series, xy and constant data
yet. ');
0080         disp('* FFT of fsdata object');
0081
0082
0083     otherwise
0084         error('### unknown data type. They can not be added. ')
0085     end
0086
0087     % add to output
0088     bs = [bs b];
0089 end
0090
0091 % Reshape the ouput to the same size of the input
0092 bs = reshape(bs, size(as));

```

```
0093
0094 varargout{1} = bs;
0095
0096
0097 %-----
0098 % Get default params
0099 function plo = getDefaultPL()
0100
0101 plo = plist();
0102
0103 % END
```


[Home](#) > [classes](#) > [@ao](#) > filter.m

filter

PURPOSE

FILTER overrides the filter function for analysis objects.

SYNOPSIS

function varargout = filter(varargin)

DESCRIPTION

FILTER overrides the filter function for analysis objects.

%%%

DESCRIPTION: FILTER overrides the filter function for analysis objects.
Applies the input digital IIR/FIR filter to the input analysis object. If the input analysis object contains a time-series (tsdata) then the filter is applied using the normal recursion algorithm. The output analysis object contains a tsdata object.

If the input analysis object contains a frequency-series (fsdata) then the response of the filter is computed and then multiplied with the input frequency series. The output analysis object contains a frequency series.

CALL:

```
>> [b, filt] = filter(a,pl)
>> [b, filt] = filter(a,filt,pl)
>> b = filter(a,pl)
```

INPUTS:

- pl - a parameter list
- a - input analysis object

OUTPUTS:

- filt - a copy of the input filter object with the history values filled in.
- b - output analysis object containing the filtered data.

PARAMETERS: filter - the filter object to use to filter the data

VERSION: \$Id: filter.m,v 1.24 2008/01/21 15:50:26 hewitson Exp \$

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

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

HISTORY: 11-02-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- **ao** AO analysis object class constructor.
- **char** CHAR overloads char() function for analysis objects.
- **filter** FILTER overrides the filter 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:

- **filter** FILTER overrides the filter function for analysis objects.

SUBFUNCTIONS

- **function pl_default = getDefaultPL()**

SOURCE CODE

```
0001 function varargout = filter(varargin)
0002 % FILTER overrides the filter function for analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: FILTER overrides the filter function for analysis objects.
0007 %               Applies the input digital IIR/FIR filter to the input analysis
0008 %               object. If the input analysis object contains a
0009 %               time-series (tsdata) then the filter is applied using the normal
0010 %               recursion algorithm. The output analysis object contains a tsdata
0011 %               object.
0012 %
0013 %               If the input analysis object contains a frequency-series (fsdata)
0014 %               then the response of the filter is computed and then multiplied
0015 %               with the input frequency series. The output analysis object
0016 %               contains a frequency series.
0017 %
0018 % CALL:         >> [b, filt] = filter(a,pl)
0019 %               >> [b, filt] = filter(a,filt,pl)
0020 %               >> b = filter(a,pl)
0021 %
```

```

0022 % INPUTS:      pl   - a parameter list
0023 %              a    - input analysis object
0024 %
0025 % OUTPUTS:      filt - a copy of the input filter object with the
0026 %                  history values filled in.
0027 %              b    - output analysis object containing the filtered data.
0028 %
0029 % PARAMETERS:   filter - the filter object to use to filter the data
0030 %
0031 % VERSION:      $Id: filter.m,v 1.24 2008/01/21 15:50:26 hewitson Exp $
0032 %
0033 % The following call returns a parameter list object that contains the
0034 % default parameter values:
0035 %
0036 % >> pl = filter(ao, 'Params')
0037 %
0038 % HISTORY: 11-02-07 M Hewitson
0039 %           Creation
0040 %
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042 %
0043 % Standard history variable
0044 ALGONAME = mfilename;
0045 VERSION  = '$Id: filter.m,v 1.24 2008/01/21 15:50:26 hewitson Exp $';
0046
0047 %% Check if this is a call for parameters
0048 if nargin == 2
0049     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0050         in = char(varargin{2});
0051         if strcmp(in, 'Params')
0052             varargout{1} = getDefaultPL();
0053             return
0054         elseif strcmp(in, 'Version')
0055             varargout{1} = VERSION;
0056             return
0057         end
0058     end
0059 end
0060
0061 %% capture input variable names
0062 fobj = [];
0063 bs   = [];
0064
0065 % Collect input ao's, plist's and ao variable names
0066 in_names = {};
0067 for ii = 1:nargin
0068     in_names{end+1} = inputname(ii);
0069
0070     if isa(varargin{ii}, 'miir') || isa(varargin{ii}, 'mfir')
0071         fobj = varargin{ii};
0072     end
0073 end
0074

```

```

0075 [as, ps, invars] = collect_inputs(varargin, in_names);
0076
0077 if isa(ps, 'plist')
0078     pl = combine(ps);
0079 else
0080     pl = plist();
0081 end
0082 if isempty(fobj)
0083     fobj = find(pl, 'filter');
0084 end
0085
0086 % check inputs
0087 if ~isa(fobj, 'miir') && ~isa(fobj, 'mfir')
0088     error('### the filter input should be an miir/mfir object.');
```

```

0089 end
0090
0091 for j=1:numel(as)
0092
0093     % get input data
0094     a = as(j);
0095     d = a.data;
0096
0097     %----- Time-series filter
0098     %
0099     if isa(d, 'tsdata')
0100
0101         % get input data
0102         fs = d.fs;
0103
0104         if isa(fobj, 'mfir')
0105
0106             % apply filter
0107             disp('* filtering with FIR filter...');
0108             G = get(fobj, 'gain');
0109
0110             % [fstruct, y] = ltpda_firfilter(struct(filt), d.y);
0111             coeffs = get(fobj, 'a');
0112             Zi      = get(fobj, 'histout');
0113             [y, Zf] = filter(coeffs, 1, G.*d.y, Zi);
0114
0115             % remove group delay
0116             if isempty(find(pl, 'gdoff'))
0117                 gd = floor(get(fobj, 'gd'));
0118                 x = d.x(1:end-gd);
0119                 y = y(1+gd:end);
0120             else
0121                 x = d.x;
0122             end
0123
0124             % consolodate this structure with mfir class before converting.
0125             fobj = set(fobj, 'histout', Zf.');
```

```

0126

```

```

0127     else %if isa(fobj, 'miir')
0128
0129         if fs ~= get(fobj, 'fs')
0130             warning('!!! Filter is designed for a different sample rate of data.');
```

% Adjust/redesign if this is a standard filter

```

0132         fobj = redesign(fobj, fs);
0133     end
0134
0135     % apply filter
0136 %     [fstruct, y] = ltpda_iirfilter(struct(fobj), d.y, length(d.y));
0137
0138     acoeffs = get(fobj, 'a');
0139     bcoeffs = get(fobj, 'b');
0140     Zi      = get(fobj, 'histout');
0141     [y, Zf] = filter(acoeffs, bcoeffs, d.y, Zi);
0142     x = d.x;
0143     % consolodate this structure with miir class before converting.
0144 %     fobj = set(fobj, 'histin', fstruct.histin);
0145 %     fobj = set(fobj, 'histout', fstruct.histout);
0146     fobj = set(fobj, 'histout', Zf);
0147 end
0148
0149 %----- Create output analysis object
0150 % make a new tsdata object
0151 ts = tsdata(y, fs);
0152 ts = set(ts, 'name', sprintf('filter %s with %s', d.name, get
(fobj, 'name')));
0153 ts = set(ts, 'xunits', d.xunits);
0154 ts = set(ts, 'yunits', d.yunits);
0155 ts = set(ts, 't0', d.t0);
0156
0157 % make a new history object
0158 pl = plist();
0159 pl = append(pl, param('filter', fobj));
0160 h = history(ALGONAME, VERSION, pl, a.hist);
0161 h = set(h, 'invars', cellstr(invars{j}));
0162
0163 % make output analysis object
0164 b = ao(ts, h);
0165 % name for this object
0166 b = setnh(b, 'name', sprintf('%s(%s)', get(fobj, 'name'), invars{j}));
0167 bs = [bs b];
0168
0169 %----- Frequency-series filter
0170 %
0171 elseif isa(d, 'fsdata')
0172     error('### I don''t work yet. Please code me up.');
```

0173

```

0174 else
0175     error('### unknown data type.');
```

0176 end

```
0177 end
0178
0179 % Reshape the ouput to the same size of the input
0180 bs = reshape(bs, size(as));
0181
0182 if nargout == 1
0183     varargout{1} = bs;
0184 elseif nargout == 2
0185     varargout{1} = bs;
0186     varargout{2} = fobj;
0187 else
0188     error('### wrong number of output arguments. ');
0189 end
0190
0191 %% Get default params
0192 function pl_default = getDefaultPL()
0193
0194     pl_default = plist(param('filter', ''));
0195     pl_default = append(pl_default, 'gdoff' , []);
```

[Home](#) > [classes](#) > [@ao](#) > filtfilt.m

filtfilt

PURPOSE

FILTFILT overrides the filtfilt function for analysis objects.

SYNOPSIS

function varargout = filtfilt(varargin)

DESCRIPTION

FILTFILT overrides the filtfilt function for analysis objects.

%%%

DESCRIPTION: FILTFILT overrides the filtfilt function for analysis objects.
Applies the input digital IIR filter to the input analysis object forwards and backwards. If the input analysis object contains a time-series (tsdata) then the filter is applied using the normal recursion algorithm. The output analysis object contains a tsdata object.

If the input analysis object contains a frequency-series (fsdata) then the response of the filter is computed and then multiplied with the input frequency series. The output analysis object contains a frequency series.

CALL: >> [b, filt] = filtfilt(a,pl)
 >> b = filtfilt(a,pl)

INPUTS: pl - a parameter list
 a - input analysis object

OUTPUTS: filt - a copy of the input filter object with the
 history values filled in.
 b - output analysis object containing the filtered data.

PARAMETERS: filter - the filter object to use to filter the data

REMARK: Uses ltpda_filtfilt() to do the filtering.

VERSION: \$Id: filtfilt.m,v 1.13 2007/11/26 14:48:13 ingo Exp \$

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

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

```
HISTORY: 11-02-07 M Hewitson
          Creation
```

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

CROSS-REFERENCE INFORMATION

This function calls:

- **ao** AO analysis object class constructor.
- **char** CHAR overloads char() function for analysis objects.
- **filtfilt** FILTFILT overrides 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.
- **set** SET set an analysis object property.
- **setnh** SET set an analysis object property without recording the history.

This function is called by:

- **filtfilt** FILTFILT overrides the filtfilt function for analysis objects.

SUBFUNCTIONS

- **function pl_default = getDefaultPL()**

SOURCE CODE

```
0001 function varargout = filtfilt(varargin)
0002 % FILTFILT overrides the filtfilt function for analysis objects.
0003 %
0004 %%%%%%%%%%
0005 %
0006 % DESCRIPTION: FILTFILT overrides the filtfilt function for analysis objects.
0007 %               Applies the input digital IIR filter to the input analysis object
0008 %               forwards and backwards. If the input analysis object contains a
0009 %               time-series (tsdata) then the filter is applied using the normal
0010 %               recursion algorithm. The output analysis object contains a tsdata
0011 %               object.
0012 %
0013 %               If the input analysis object contains a frequency-series (fsdata)
0014 %               then the response of the filter is computed and then multiplied
0015 %               with the input frequency series. The output analysis object
0016 %               contains a frequency series.
0017 %
0018 % CALL:         >> [b, filt] = filtfilt(a,pl)
0019 %               >> b = filtfilt(a,pl)
0020 %
```



```

0021 % INPUTS:      pl   - a parameter list
0022 %              a    - input analysis object
0023 %
0024 % OUTPUTS:      filt - a copy of the input filter object with the
0025 %                  history values filled in.
0026 %              b    - output analysis object containing the filtered data.
0027 %
0028 % PARAMETERS:   filter - the filter object to use to filter the data
0029 %
0030 % REMARK:        Uses ltpda_filtfilt() to do the filtering.
0031 %
0032 % VERSION:       $Id: filtfilt.m,v 1.13 2007/11/26 14:48:13 ingo Exp $
0033 %
0034 % The following call returns a parameter list object that contains the
0035 % default parameter values:
0036 %
0037 % >> pl = filtfilt(ao, 'Params')
0038 %
0039 % HISTORY: 11-02-07 M Hewitson
0040 %           Creation
0041 %
0042 %%%%%%%%%%%
0043
0044 ALGONAME = mfilename;
0045 VERSION  = '$Id: filtfilt.m,v 1.13 2007/11/26 14:48:13 ingo Exp $';
0046 bo       = [];
0047 filt     = [];
0048
0049 %% Check if this is a call for parameters
0050 if nargin == 2
0051     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0052         in = char(varargin{2});
0053         if strcmp(in, 'Params')
0054             varargout{1} = getDefaultPL();
0055             return
0056         elseif strcmp(in, 'Version')
0057             varargout{1} = VERSION;
0058             return
0059         end
0060     end
0061 end
0062
0063 % Collect input ao's, plist's and ao variable names
0064 in_names = {};
0065 for ii = 1:nargin
0066     in_names{end+1} = inputname(ii);
0067
0068     if isa(varargin{ii}, 'miir')
0069         filt = varargin{ii};
0070     end
0071 end
0072
0073 [as, ps, invars] = collect_inputs(varargin, in_names);

```

```

0074
0075 if isa(ps, 'plist')
0076     pl = combine(ps);
0077 else
0078     pl = plist();
0079 end
0080
0081 if isempty(filt)
0082     filt = find(pl, 'filter');
0083 end
0084
0085 % check inputs
0086 if ~isa(filt, 'miir')
0087     error('### the first input should be an miir object.');
```

```

0088 end
0089 if ~isa(as, 'ao')
0090     error('### second input should be an analysis object.');
```

```

0091 end
0092
0093 for ii=1:numel(as)
0094
0095     % get input data
0096     a = as(ii);
0097     d = a.data;
0098
0099     %----- Time-series filter
0100     %
0101     if isa(d, 'tsdata')
0102
0103         % get input data
0104         y = d.y;
0105         fs = d.fs;
0106         if fs ~= get(filt, 'fs')
0107             warning('!!! Filter is designed for a different sample rate of data.');
```

```

0108             % Adjust/redesign if this is a standard filter
0109             filt = redesign(filt, fs);
0110         end
0111
0112         % get filter coeffs
0113         ac = get(filt, 'a');
0114         bc = get(filt, 'b');
```

```

0115
0116         % apply filter
0117         y = filtfilt(ac, bc, y);
0118         % [fstruct, y] = ltpda_filtfilt(struct(filt), d.y, length(d.y));
0119         % consolodate this structure with miir class before converting.
0120         % filt = set(filt, 'histin', fstruct.histin);
0121         % filt = set(filt, 'histout', fstruct.histout);
0122
0123         %----- Create output analysis object
0124         % make a new tsdata object
0125         ts = tsdata(d.x, y);

```

```

0126     ts = set(ts, 'name', sprintf('filtfilt %s with %s', d.name, get
(filt, 'name')));
0127     ts = set(ts, 'xunits', d.xunits);
0128     ts = set(ts, 'yunits', d.yunits);
0129     ts = set(ts, 't0', d.t0);
0130
0131     % make a new history object
0132     pl = plist();
0133     pl = append(pl, param('filter', filt));
0134     h = history(ALGONAME, VERSION, pl, a.hist);
0135     h = set(h, 'invars', cellstr(invars{ii}));
0136
0137     % make output analysis object
0138     bs = ao(ts, h);
0139     % name for this object
0140     bs = setnh(bs, 'name', sprintf('%s(%s)', get(filt, 'name'), invars{ii}));
0141
0142     bo = [bo bs];
0143
0144     %----- Frequency-series filter
0145     %
0146     elseif isa(d, 'fsdata')
0147         error('### I don''t work yet. Please code me up.');
```

```

0148
0149     else
0150         error('### unknown data type.');
```

```

0151     end
0152
0153 end
0154
0155 % Reshape the output to the same size of the input
0156 bo = reshape(bo, size(as));
0157
0158 if nargout == 1
0159     varargout{1} = bo;
0160 elseif nargout == 2
0161     varargout{1} = bo;
0162     varargout{2} = filt;
0163 else
0164     error('### wrong number of output arguments.');
```

```

0165 end
0166
0167
0168 %% Get default params
0169 function pl_default = getDefaultPL()
0170
0171     pl_default = plist(param('filter', ''));

```

get

PURPOSE

GET get ao properties.

SYNOPSIS

function val = get(ao, propName)

DESCRIPTION

GET get ao properties.

%%%

DESCRIPTION: GET get ao properties.

CALL: name = get(ao, 'name');
provenance = get(ao, 'provenance');
description = get(ao, 'description');
history = get(ao, 'hist');
data = get(ao, 'data');
mfile = get(ao, 'mfile');
amfilename = get(ao, 'mfilename');
mdlfile = get(ao, 'mdlfile');
mdlfilename = get(ao, 'mdlfilename');
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 = ao(ao, 'Params')

VERSION: \$Id: get.m,v 1.10 2007/11/14 16:30:18 ingo Exp \$

HISTORY: 30-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- **ao** AO analysis object class constructor.
- **char** CHAR overloads char() function for analysis objects.
- **get** GET get ao properties.

This function is called by:

- **ctranspose** CTRANSPOSE overloads the ' operator for Analysis Objects.
- **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.
- **eig** EIG overloads the determinant function for Analysis objects.
- **fft** FFT overloads the fft operator for Analysis objects.
- **filter** FILTER overrides the filter function for analysis objects.
- **filtfilt** FILTFILT overrides the filtfilt function for analysis objects.
- **get** GET get ao properties.
- **index** INDEX index into an AO array or matrix. This properly captures the history.
- **inv** INV overloads the inverse function for Analysis Objects.
- **iplot** IPLOT provides an intelligent plotting tool for LTPDA.
- **norm** NORM overloads the norm operator for Analysis Objects.
- **plot** PLOT a simple plot of analysis objects.
- **simple_plot** PLOT a simple plot of analysis objects.
- **string** STRING writes a command string that can be used to recreate the input Analysis object(s).
- **submit_fastinsert** SUBMIT AO(s) to the MySQL server.
- **svd** SVD overloads the determinant function for Analysis objects.
- **transpose** TRANSPOSE overloads the .' operator for Analysis Objects.
- **xmladd** XMLADD add an analysis object to the given node of the input XML DOM

SUBFUNCTIONS

- **function plo = getDefaultPL()**

SOURCE CODE

```

0001 function val = get(ao, propName)
0002 % GET get ao properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get ao properties.
0007 %
0008 % CALL: name          = get(ao, 'name');
0009 %      provenance     = get(ao, 'provenance');
0010 %      description    = get(ao, 'description');
0011 %      history        = get(ao, 'hist');
0012 %      data           = get(ao, 'data');

```

```

0013 %         mfile         = get(ao, 'mfile');
0014 %         amfilename    = get(ao, 'mfilename');
0015 %         mdlfile       = get(ao, 'mdlfile');
0016 %         mdlfilename    = get(ao, 'mdlfilename');
0017 %         version       = get(ao, 'version');
0018 %         created       = get(ao, 'created');
0019 %         fs            = get(ao, 'fs');           --> ao.data.fs;
0020 %         t0           = get(ao, 't0');           --> ao.data.t0;
0021 %
0022 % The following call returns a parameter list object that contains the
0023 % default parameter values:
0024 %
0025 % >> pl = ao(ao, 'Params')
0026 %
0027 % VERSION: $Id: get.m,v 1.10 2007/11/14 16:30:18 ingo Exp $
0028 %
0029 % HISTORY: 30-01-07 M Hewitson
0030 %             Creation
0031 %
0032 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0033
0034 VERSION = '$Id: get.m,v 1.10 2007/11/14 16:30:18 ingo Exp $';
0035
0036 % Check if this is a call for parameters or for the cvs-version number
0037 if nargin == 2
0038     if isa(ao, 'ao') && ischar(propName)
0039         in = char(propName);
0040         if strcmp(in, 'Params')
0041             val = getDefaultPL();
0042             return
0043         elseif strcmp(in, 'Version')
0044             val = VERSION;
0045             return
0046         end
0047     end
0048 end
0049
0050 fields = fieldnames(ao);
0051
0052 if ~ismember(propName, [fields; 'fs'; 't0'])
0053     error('### '' propName, '' is not a valid analysis object property.');
```

```

0054 else
0055     %%% Special case: If possible return the 'fs' property of the data object
0056     if strcmp(propName, 'fs')
0057         if isa(ao.data, 'tsdata') || isa(ao.data, 'fsdata')
0058             val = ao.data.fs;
0059         else
0060             error('### No ''fs'' property for this data type');
0061         end
0062     end
0063
0064     %%% Special case: If possible return the 't0' property of the data object

```

```
0065     elseif strcmp(propName, 't0')
0066
0067         if isa(ao.data, 'tsdata')
0068             val = get(ao.data, 't0');
0069         else
0070             error('### The data object is not from the class tsdata');
0071         end
0072
0073         %% Return the property of the analysis object
0074     else
0075         val = ao.(propName);
0076     end
0077 end
0078
0079
0080
0081 % Get default params
0082 function plo = getDefaultPL()
0083
0084 plo = plist();
```

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')

VERSION: \$Id: imag.m,v 1.4 2007/11/26 14:48:13 ingo Exp \$

HISTORY: 20-0802007 Diepholz
 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:

- **export** EXPORT export an analysis object to a text file.
- **ipLOT** IPLOT provides an intelligent plotting tool for LTPDA.
- **plot** PLOT a simple plot of analysis objects.
- **plot_old** PLOT plots an array of analysis objects.
- **simple_plot** PLOT a simple plot of analysis objects.

SUBFUNCTIONS

- **function pl_default = getDefaultPL()**

SOURCE CODE

```

0001 function ao_out = imag(varargin)
0002 % IMAG overloads the imaginary operator for Analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: IMAG overloads the imaginary operator for Analysis objects.
0007 %
0008 % CALL: ao_out = imag(ao_in);
0009 %         ao_out = imag(ao_in, pl);
0010 %
0011 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0012 %                   ao_in  = ao_vector
0013 %                   ao_in  = ao_matrix
0014 %
0015 % PARAMETER LIST:  <key>          <value>          <description>
0016 %                   tsdata    fsdata    xydata
0017 %                   'xdata'    't'       'f'       'x'       compute the xdata
0018 %                   'ydata'    'x'       'xx'      'y'       compute the ydata
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = imag(ao, 'Params')
0024 %
0025 % VERSION: $Id: imag.m,v 1.4 2007/11/26 14:48:13 ingo Exp $
0026 %
0027 % HISTORY: 20-0802007 Diepholz
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

0031
0032 VERSION = '$Id: imag.m,v 1.4 2007/11/26 14:48:13 ingo Exp $';
0033 ao_out = [];
0034
0035 %% Check if this is a call for parameters
0036 if nargin == 2
0037     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0038         in = char(varargin{2});
0039         if strcmp(in, 'Params')
0040             ao_out = getDefaultPL();
0041             return
0042         elseif strcmp(in, 'Version')
0043             ao_out = VERSION;
0044             return
0045         end
0046     end
0047 end
0048
0049 %% Collect input ao's, plist's and ao variable names
0050 in_names = {};
0051 for ii = 1:nargin
0052     in_names{end+1} = inputname(ii);
0053 end
0054
0055 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0056
0057 if ~isempty(pl)
0058     pl = combine(pl);
0059 end
0060
0061 %% go through analysis objects
0062 for j=1:numel(ao_set)
0063     a = ao_set(j);
0064
0065     [h, imag_data] = single_operation(a.data, 'imag', pl);
0066
0067     %% Add the history from the ao object to the history
0068     h = set(h, 'inhists', [a.hist]);
0069
0070     %% Set the var_name to the history
0071     h = set(h, 'invars', cellstr(invars{j}));
0072
0073     %% create a new analysis objects
0074     new_ao = ao (imag_data, h);
0075     new_ao = setnh(new_ao, 'name', sprintf('imag(%s)', invars{j}) );
0076
0077     ao_out = [ao_out new_ao];
0078
0079 end
0080
0081 % Reshape the ouput to the same size of the input

```

```
0082 ao_out = reshape(ao_out, size(ao_set));
0083
0084
0085 %% Get default params
0086 function pl_default = getDefaultPL()
0087
0088     pl_default = plist([param('xdata', '')
0089                          param('ydata', '')]);
0090
0091
0092 % END
```

[Home](#) > [classes](#) > [@ao](#) > interp.m

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 1.10 2007/11/26 14:48:13 ingo Exp \$

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

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

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.
- **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

```

0001 function varargout = interp(varargin)
0002 % INTERP interpolate the values in the input AO(s) at new values specified by
the input parameter list.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: INTERP interpolate the values in the input AO(s) at new values
0007 %               specified by the input parameter list.
0008 %
0009 % CALL:         b = interp(a, pl)
0010 %
0011 % INPUTS:       a - input array of AOs
0012 %               pl - parameter list with the keys 'vertices' and 'method'
0013 %
0014 % OUTPUTS:      b - output array of AOs
0015 %
0016 % PARAMETERS:   'vertices' - new set of vertices to interpolate on.
0017 %               'method'   - one of:
0018 %                           'nearest' - nearest neighbour
0019 %                           'linear'  - linear interpolation
0020 %                           'spline'  - spline interpolation
0021 %                           'cubic'   - shape-preserving piecewise cubic interpolation
0022 %
0023 % REMARK:       Matrix cdata objects are not supported.
0024 %
0025 % VERSION:      $Id: interp.m,v 1.10 2007/11/26 14:48:13 ingo Exp $
0026 %
0027 % The following call returns a parameter list object that contains the
0028 % default parameter values:

```

```

0029 %
0030 % >> pl = interp(ao, 'Params')
0031 %
0032 % HISTORY: 05-06-07 M Hewitson
0033 %             Creation
0034 %
0035 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0036
0037 ALGONAME = mfilename;
0038 VERSION  = '$Id: interp.m,v 1.10 2007/11/26 14:48:13 ingo Exp $';
0039
0040 % Check if this is a call for parameters
0041 if nargin == 2
0042     in = char(varargin{2});
0043     if strcmp(in, 'Params')
0044         varargout{1} = getDefaultPL();
0045         return
0046     elseif strcmp(in, 'Version')
0047         varargout{1} = VERSION;
0048         return
0049     end
0050 end
0051
0052 % Collect input ao's, plist's and ao variable names
0053 in_names = {};
0054 for ii = 1:nargin
0055     in_names{end+1} = inputname(ii);
0056 end
0057
0058 [as, ps, invars] = collect_inputs(varargin, in_names);
0059
0060 % check plist
0061 if isempty(ps)
0062     pl = getDefaultPL();
0063 else
0064     pl = combine(ps, getDefaultPL);
0065 end
0066
0067
0068 % Get parameters
0069 vertices = find(pl, 'vertices');
0070 method   = find(pl, 'method');
0071
0072 %-----
0073 % Loop over input AOs
0074 bs = [];
0075 for j=1:numel(as)
0076
0077     a = as(j);
0078
0079     %-----

```

```

0080 % Get data
0081 [x,y] = get_xy_values(a.data);
0082
0083 %-----
0084 % Interpolate this vector
0085 if ~isempty(x)
0086     % for tsdata, fsdata and xydata objects
0087     y = interp1(x,y,vertices, method, 'extrap');
0088 else
0089     % for cdata object
0090     y = interp1(y,vertices, method, 'extrap');
0091 end
0092
0093 %-----
0094 % Make output A0/cdata
0095 d = a.data;
0096
0097 d = set_xy_axis(d, vertices, y.');
0098
0099 d = set(d, 'name', sprintf('%s(%s)', method, d.name));
0100
0101 h = history(ALGONAME, VERSION, pl, a.hist);
0102 h = set(h, 'invars', invars(j));
0103
0104 % make output analysis object
0105 b = ao(d, h);
0106
0107 % set name
0108 b = setnh(b, 'name', sprintf('%s(%s)', method, invars{j}));
0109
0110 % add to outputs
0111 bs = [bs b];
0112
0113 end
0114
0115 % Reshape the ouput to the same size of the input
0116 bs = reshape(bs, size(as));
0117
0118 % Set outputs
0119 varargout{1} = bs;
0120
0121 %-----
0122 % Get default params
0123 function plo = getDefaultPL()
0124
0125 plo = plist();
0126 plo = append(plo, param('vertices', []));
0127 plo = append(plo, param('method', 'spline'));
0128
0129
0130
0131

```

```
0132
```

```
0133 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > inv.m

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 1.14 2008/01/20 12:53:30 hewitson Exp \$

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

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

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

```

0001 function varargout = inv(varargin)
0002 % INV overloads the inverse function for Analysis Objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: INV overloads the inverse function for Analysis Objects.
0007 %
0008 % CALL:          a = inv(a1)    % only with data = cdata
0009 %
0010 % VERSION:       $Id: inv.m,v 1.14 2008/01/20 12:53:30 hewitson Exp $
0011 %
0012 % The following call returns a parameter list object that contains the
0013 % default parameter values:
0014 %
0015 % >> pl = inv(ao, 'Params')
0016 %
0017 % HISTORY: 08-05-07 A Monsky
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: inv.m,v 1.14 2008/01/20 12:53:30 hewitson Exp $';
0023 bs      = [];
0024
0025 %% Check if this is a call for parameters
0026
0027 if nargin == 2
0028     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0029         in = char(varargin{2});
0030         if strcmp(in, 'Params')
0031             varargout{1} = getDefaultPL();
0032             return
0033         elseif strcmp(in, 'Version')
0034             varargout{1} = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 % Collect input ao's, plist's and ao variable names
0041 in_names = {};
0042 for ii = 1:nargin
0043     in_names{end+1} = inputname(ii);
0044 end

```

```

0045
0046 [as, ps, invars] = collect_inputs(varargin, in_names);
0047
0048 % check plist
0049 if isempty(ps)
0050     pl = getDefaultPL();
0051 else
0052     pl = combine(ps, getDefaultPL);
0053 end
0054
0055 %% go through analysis objects
0056 for j=1:numel(as)
0057     a = as(j);
0058     d = get(a, 'data');
0059     dinfo = whos('d');
0060
0061     % Which data type do we have
0062     dtype = dinfo.class;
0063     switch dtype
0064     case 'cdata'
0065         if size(d.y,1) ~= size(d.y,2)
0066             error('### The value (y) of data must be a square matrix.')
0067         else
0068             %disp('* INVERSE of cdata object');
0069
0070             % make a new cdata object
0071             [h, c] = single_operation(d, 'inv', pl);
0072
0073             % make output analysis object
0074             h = set(h, 'inhists', [a.hist]);
0075
0076             %% Set the var_name to the history
0077             h = set(h, 'invars', cellstr(invars{j}));
0078
0079             b = ao(c, h);
0080             b = setnh(b, 'name', sprintf('inv(%s)', char(invars{j})));
0081         end
0082
0083     case {'tsdata', 'fsdata', 'xydata'}
0084         error('### this function works for cdata type A0 only')
0085     otherwise
0086         error('### unknown data type.')
0087     end
0088 end
0089
0090 % add to output
0091 bs = [bs b];
0092 end
0093
0094 % Reshape the ouput to the same size of the input
0095 bs = reshape(bs, size(as));

```

```
0096 varargout{1} = bs;  
0097  
0098 %% Get default params  
0099 function plo = getDefaultPL()  
0100  
0101 plo = plist();  
0102  
0103 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > join.m

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 1.11 2007/11/26 14:48:13 ingo 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')
```

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

```

0001 function varargout = join(varargin)
0002 % JOIN multiple AOs into a single AO.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: JOIN multiple AOs into a single AO.
0007 %             If any two AOs overlap, then the values from the first appear
0008 %             in the output AO.
0009 %
0010 % CALL:       b = join(a1,a2,pl)
0011 %
0012 % VERSION:    $Id: join.m,v 1.11 2007/11/26 14:48:13 ingo Exp $
0013 %
0014 % REMARK:     Input AOs should be of the same type; if not, only AOs of the
0015 %             type of the first input AO will be joined together to produce
0016 %             the output.
0017 %
0018 % The following call returns a parameter list object that contains the
0019 % default parameter values:
0020 %
0021 % >> pl = join(ao, 'Params')
0022 %
0023 % HISTORY: 03-06-07 M Hewitson
0024 %             Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0027
0028 ALGONAME = mfilename;
0029 VERSION  = '$Id: join.m,v 1.11 2007/11/26 14:48:13 ingo Exp $';
0030
0031 %% Check if this is a call for parameters
0032 if nargin == 2
0033     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0034         in = char(varargin{2});
0035         if strcmp(in, 'Params')
0036             varargout{1} = getDefaultPL();
0037             return
0038         elseif strcmp(in, 'Version')
0039             varargout{1} = VERSION;

```

```

0040         return
0041     end
0042 end
0043 end
0044
0045
0046 % Collect input ao's, plist's and ao variable names
0047 in_names = {};
0048 for ii = 1:nargin
0049     in_names{end+1} = inputname(ii);
0050 end
0051
0052 [as, ps, invars] = collect_inputs(varargin, in_names);
0053
0054 % check plist
0055 if isempty(ps)
0056     pl = getDefaultPL();
0057 else
0058     pl = combine(ps, getDefaultPL);
0059 end
0060
0061 %-----
0062 % Get data type from the first AO
0063 a      = as(1).data;
0064 dinfo = whos('a');
0065 dtype = dinfo.class;
0066
0067 %-----
0068 % Go through each AO and collect the data of type 'dtype'
0069
0070 histin = [];
0071 xo      = [];
0072 yo      = [];
0073 fs      = -1;
0074 dname   = [];
0075 xunits  = '';
0076 yunits  = '';
0077
0078 % Compute time offset for tsdata objects
0079 % to avoid rounding errors later
0080
0081 Toff = getToff(as);
0082
0083 % loop over AOs
0084 for j=1:numel(as)
0085
0086     a = as(j);
0087
0088     % Only get the data type we want
0089     if isa(a.data, dtype)
0090
0091         switch dtype

```

```

0092     case 'tsdata'
0093
0094         % here we concatonate time-series
0095         [x,y] = get_xy_values(a.data);
0096         t0 = (a.data.t0.utc_epoch_milli - Toff)/1000;
0097
0098         % make proper time vector
0099         x = x + t0;
0100
0101         % only add samples past the end of existing
0102         if isempty(xo)
0103             yo = y;
0104             xo = x;
0105         else
0106             idxPost = find(x > max(xo));
0107             idxPre = find(x < min(xo));
0108             xo = [x(idxPre);xo;x(idxPost)];
0109             yo = [y(idxPre);yo;y(idxPost)];
0110         end
0111
0112         % store fs
0113         if fs>0 && a.data.fs ~= fs
0114             warning('!!! Data has different sample rates !!!');
0115         end
0116         fs = a.data.fs;
0117
0118         % store xunits
0119         if ~strcmp(xunits, a.data.xunits) && ~isempty(xunits)
0120             warning('!!! Data has different X units !!!');
0121         end
0122         xunits = a.data.xunits;
0123
0124         % store yunits
0125         if ~strcmp(yunits, a.data.yunits) && ~isempty(yunits)
0126             warning('!!! Data has different Y units !!!');
0127         end
0128         yunits = a.data.yunits;
0129
0130     case 'fsdata'
0131         error('### I only work with time-series data at the moment.');
```

```

0132     case 'xydata'
0133         error('### I only work with time-series data at the moment.');
```

```

0134     case 'cdata'
0135         error('### I only work with time-series data at the moment.');
```

```

0136     otherwise
0137         error('### Unknown data type');
```

```

0138 end
0139
0140 % Collect this input history
0141 histin = [histin a.hist];
0142
0143 % Collect names, invars

```



```

0144     dname = [dname ', ' a.data.name];
0145
0146     else
0147         warning('!!! Ignoring AO input with data type %s', dtype);
0148     end
0149
0150 end
0151 % Tidy up names
0152 dname = dname(2:end);
0153
0154 %-----
0155 % Now sort output vectors
0156 [xos, idx] = sort(xo);
0157 yos = yo(idx);
0158
0159 %% Build output data object
0160 switch dtype
0161     case 'tsdata'
0162
0163         % get t0
0164         t0 = xos(1);
0165         xos = xos - t0;
0166         data = tsdata(xos, yos);
0167         data = set(data, 't0', (Toff+t0));
0168         data = set(data, 'name', dname);
0169         data = set(data, 'xunits', xunits);
0170         data = set(data, 'yunits', yunits);
0171
0172     case 'fsdata'
0173
0174     case 'xydata'
0175
0176     case 'cdata'
0177
0178 end
0179
0180 %-----
0181 % Build output AO
0182
0183 h = history(ALGONAME, VERSION, pl, histin);
0184 h = set(h, 'invars', invars);
0185 a = ao(data, h);
0186
0187 %-----
0188 % set output
0189
0190 varargout{1} = a;
0191
0192
0193 %-----
0194 % Get Offset of this set of time-vectors

```

```
0195 function Toff = getToff(as)
0196
0197 Toff = 1e20;
0198 for j=1:length(as)
0199     if isa(as(j).data, 'tsdata')
0200         t0 = as(j).data.t0.utc_epoch_milli;
0201         if t0 < Toff
0202             Toff = t0;
0203         end
0204     end
0205 end
0206
0207 %-----
0208 % Get default params
0209 function plo = getDefaultPL()
0210
0211 plo = plist();
0212
0213
0214
0215
0216
0217 % END
```

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')

HISTORY: 23-05-2007 Diepholz
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:

- **ln** LN overloads the log operator for Analysis objects. Natural logarithm.

SUBFUNCTIONS

- **function pl_default = getDefaultPL()**

SOURCE CODE

```

0001 function ao_out = log (varargin)
0002 % LOG overloads the log operator for Analysis objects. Natural logarithm.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LOG overloads the log operator for Analysis objects.
0007 %               Natural logarithm.
0008 %               LOG(ao) is the natural logarithm of the elements of ao.data.
0009 %
0010 % CALL: ao_out = log(ao_in);
0011 %       ao_out = log(ao_in, pl);
0012 %       ao_out = log(ao1, pl1, ao_vector, ao_matrix, pl2);
0013 %
0014 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0015 %                   ao_in  = ao_vector
0016 %                   ao_in  = ao_matrix
0017 %
0018 % PARAMETER LIST:  <key>          <value>          <description>
0019 %                   tsdata  fsdata  xydata
0020 %                   'xdata'  't'     'f'     'x'     compute the xdata
0021 %                   'ydata'  'x'     'xx'    'y'     compute the ydata
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 % >> pl = log(ao, 'Params')
0027 %
0028 % HISTORY: 23-05-2007 Diepholz
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032 %
0033 VERSION    = '$Id: log.m,v 1.12 2007/11/26 14:48:13 ingo Exp $';

```

```

0034 ao_out    = [];
0035
0036 %% Check if this is a call for parameters
0037 if nargin == 2
0038     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0039         in = char(varargin{2});
0040         if strcmp(in, 'Params')
0041             ao_out = getDefaultPL();
0042             return
0043         elseif strcmp(in, 'Version')
0044             ao_out = VERSION;
0045             return
0046         end
0047     end
0048 end
0049
0050 %% Collect input ao's, plist's and ao variable names
0051 in_names = {};
0052 for ii = 1:nargin
0053     in_names{end+1} = inputname(ii);
0054 end
0055
0056 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0057
0058 if ~isempty(pl)
0059     pl = combine(pl);
0060 end
0061
0062 %% go through analysis objects
0063 for j=1:numel(ao_set)
0064     a = ao_set(j);
0065
0066     [h, log_data] = single_operation(a.data, 'log', pl);
0067
0068     %% Add the history from the ao object to the history
0069     h = set(h, 'inhists', [a.hist]);
0070
0071     %% Set the var_name to the history
0072     h = set(h, 'invars', cellstr(invars{j}));
0073
0074     %% create a new analysis objects
0075     new_ao = ao (log_data, h);
0076     new_ao = setnh(new_ao, 'name', sprintf('log(%s)', invars{j}));
0077
0078     ao_out = [ao_out new_ao];
0079
0080 end
0081
0082 % Reshape the output to the same size of the input
0083 ao_out = reshape(ao_out, size(ao_set));
0084

```

```
0085 %% Get default params
0086 function pl_default = getDefaultPL()
0087
0088     pl_default = plist([param('xdata', '')
0089                          param('ydata', '')] );
0090
0091 % END
```

[Home](#) > [classes](#) > [@ao](#) > log10.m

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')
```

HISTORY: 23-05-2007 Diepholz
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

```

0001 function ao_out = log10 (varargin)
0002 % LOG10 overloads the log10 operator for Analysis objects. Common (base 10)
    logarithm.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LOG10 overloads the log10 operator for Analysis objects.
0007 %               Common (base 10) logarithm.
0008 %               LOG10(ao) is the base 10 logarithm of the elements of ao.data.
0009 %
0010 % CALL: ao_out = log10(ao_in);
0011 %       ao_out = log10(ao_in, pl);
0012 %       ao_out = log10(ao1, pl1, ao_vector, ao_matrix, pl2);
0013 %
0014 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0015 %                   ao_in  = ao_vector
0016 %                   ao_in  = ao_matrix
0017 %
0018 % PARAMETER LIST:  <key>          <value>          <description>
0019 %                   tsdata  fsdata  xydata
0020 %                   'xdata'  't'      'f'      'x'      compute the xdata
0021 %                   'xdata'  'x'      'xx'     'y'      compute the ydata
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 % >> pl = log10(ao, 'Params')
0027 %
0028 % HISTORY: 23-05-2007 Diepholz
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032 %
0033 VERSION  = '$Id: log10.m,v 1.9 2007/11/26 14:48:13 ingo Exp $';
0034 ao_out   = [];
0035
0036 %% Check if this is a call for parameters

```



```

0037 if nargin == 2
0038     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0039         in = char(varargin{2});
0040         if strcmp(in, 'Params')
0041             ao_out = getDefaultPL();
0042             return
0043         elseif strcmp(in, 'Version')
0044             ao_out = VERSION;
0045             return
0046         end
0047     end
0048 end
0049
0050 %% Collect input ao's, plist's and ao variable names
0051 in_names = {};
0052 for ii = 1:nargin
0053     in_names{end+1} = inputname(ii);
0054 end
0055
0056 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0057
0058 if ~isempty (pl)
0059     pl = combine(pl);
0060 end
0061
0062 %% go through analysis objects
0063 for j=1:numel(ao_set)
0064     a = ao_set(j);
0065
0066     [h, log10_data] = single_operation(a.data, 'log10', pl);
0067
0068     %% Add the history from the ao object to the history
0069     h = set(h, 'inhists', [a.hist]);
0070
0071     %% Set the var_name to the history
0072     h = set(h, 'invars', cellstr(invars{j}));
0073
0074     %% create a new analysis objects
0075     new_ao = a;
0076     new_ao = ao (log10_data, h);
0077     new_ao = setnh(new_ao, 'name', sprintf('log10(%s)', invars{j})) );
0078
0079     ao_out = [ao_out new_ao];
0080
0081 end
0082
0083 % Reshape the output to the same size of the input
0084 ao_out = reshape(ao_out, size(ao_set));
0085
0086 %% Get default params
0087 function pl_default = getDefaultPL()
0088

```

```
0089    pl_default = plist([param('xdata', '')
0090                          param('ydata', '']));
0091
0092 % END
```

Generated on Tue 22-Jan-2008 10:39:13 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.

%%%

DESCRIPTION: MEAN overloads the mean operator for Analysis objects.
Compute the mean value.
MEAN(ao) is the mean value of the elements in ao.data.

CALL: ao_out = mean(ao_in);
ao_out = mean(ao_in, dim);
ao_out = mean(ao_in, pl);
ao_out = mean(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 mean along the dimension dim
		tsdata fsdata xydata	
	'xdata'	't' 'f' 'x'	compute the mean of the x-axis
	'ydata'	'x' 'xx' 'y'	compute the mean of the y-axis

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

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

HISTORY: 23-05-2007 Diepholz
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 pl_default = getDefaultPL\(\)](#)

SOURCE CODE

```

0001 function ao_out = mean (varargin)
0002 % MEAN overloads the mean operator for Analysis objects. Compute the mean value.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MEAN overloads the mean operator for Analysis objects.
0007 %               Compute the mean value.
0008 %               MEAN(ao) is the mean value of the elements in ao.data.
0009 %
0010 % CALL: ao_out = mean(ao_in);
0011 %       ao_out = mean(ao_in, dim);
0012 %       ao_out = mean(ao_in, pl);
0013 %       ao_out = mean(ao1, pl1, ao_vector, ao_matrix, pl2);
0014 %
0015 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0016 %                   ao_in  = ao_vector
0017 %                   ao_in  = ao_matrix
0018 %
0019 % PARAMETER LIST:  <key>          <value>          <description>
0020 %                  'dim'          1 or 2 or 3 ...    takes the mean along the
0021 %                                     dimension dim
0022 %
0023 %                  tsdata  fsdata  xydata
0024 %                  'xdata' 't'     'f'     'x'     compute the mean of the x-
axis
0025 %                  'ydata' 'x'     'xx'    'y'     compute the mean of the y-
axis
0026 %
0027 % The following call returns a parameter list object that contains the
0028 % default parameter values:
0029 % >> pl = mean(ao, 'Params')
0030 %

```

```

0031 % HISTORY: 23-05-2007 Diepholz
0032 %           Creation
0033 %
0034 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0035
0036 VERSION = '$Id: mean.m,v 1.12 2007/11/26 14:48:13 ingo Exp $';
0037 ao_out   = [];
0038
0039 %% Check if this is a call for parameters
0040 if nargin == 2
0041     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0042         in = char(varargin{2});
0043         if strcmp(in, 'Params')
0044             ao_out = getDefaultPL();
0045             return
0046         elseif strcmp(in, 'Version')
0047             ao_out = VERSION;
0048             return
0049         end
0050     end
0051 end
0052
0053 %% Collect input ao's, plist's and ao variable names
0054 in_names = {};
0055 ps       = plist();
0056 for ii = 1:nargin
0057     in_names{end+1} = inputname(ii);
0058
0059     if isnumeric(varargin{ii})
0060         ps = [ps plist(param('dim', varargin{ii}))];
0061     end
0062 end
0063
0064 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0065
0066 pl = combine(pl, ps);
0067
0068 %% go through analysis objects
0069 for j=1:numel(ao_set)
0070     a = ao_set(j);
0071
0072     [h, mean_data] = single_operation(a.data, 'mean', pl);
0073
0074     %% Add the history from the ao object to the history
0075     h = set(h, 'inhists', [a.hist]);
0076
0077     %% Set the var_name to the history
0078     h = set(h, 'invars', cellstr(invars{j}));
0079
0080     % convert to cdata type
0081     mean_x   = [];
0082     mean_y   = [];

```

```

0083
0084 do_xdata = find(pl, 'xdata');
0085 do_ydata = find(pl, 'ydata');
0086
0087 %% Is no axis entry in the parameter list
0088 %% then set the default axis = xdata
0089 if isempty(do_xdata) && isempty(do_ydata)
0090     do_ydata = 'yes';
0091 end
0092
0093 %% Get the operation result. It is stored in the individual axis
0094 pl_get_axis = plist([param('xdata', do_xdata) param('ydata', do_ydata)]);
0095 [mean_x, mean_y] = get_xy_values(mean_data, pl_get_axis);
0096
0097 mean_cdata = cdata([mean_x mean_y]);
0098 mean_cdata = set (mean_cdata, 'name', mean_data.name);
0099 mean_cdata = set (mean_cdata, 'xunits', mean_data.xunits);
0100 mean_cdata = set (mean_cdata, 'yunits', mean_data.yunits);
0101
0102 %% create a new analysis objects
0103 new_ao = ao (mean_cdata, h);
0104 new_ao = setnh(new_ao, 'name', sprintf('mean(%s)', invars{j}) );
0105
0106 ao_out = [ao_out new_ao];
0107
0108 end
0109
0110 % Reshape the ouput to the same size of the input
0111 ao_out = reshape(ao_out, size(ao_set));
0112
0113
0114 %% Get default params
0115 function pl_default = getDefaultPL()
0116
0117     pl_default = plist([param('dim',    '')
0118                         param('xdata',  '')
0119                         param('ydata',  '')] );
0120
0121 % END

```

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.

%%%

DESCRIPTION: MEDIAN overloads the median operator for Analysis objects.
Compute the median value.
MEDIAN(ao) is the median value of the elements in ao.data.

CALL: ao_out = median(ao_in);
ao_out = median(ao_in, dim);
ao_out = median(ao_in, pl);
ao_out = median(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 median along the dimension dim
		tsdata fsdata xydata	
	'xdata'	't' 'f' 'x'	compute the median of the x-axis
	'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')

HISTORY: 23-05-2007 Diepholz
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 pl_default = getDefaultPL\(\)](#)

SOURCE CODE

```

0001 function ao_out = median (varargin)
0002 % MEDIAN overloads the median operator for Analysis objects. Compute the median
0003 % value.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MEDIAN overloads the median operator for Analysis objects.
0007 %               Compute the median value.
0008 %               MEDIAN(ao) is the median value of the elements in ao.data.
0009 %
0010 % CALL: ao_out = median(ao_in);
0011 %       ao_out = median(ao_in, dim);
0012 %       ao_out = median(ao_in, pl);
0013 %       ao_out = median(ao1, pl1, ao_vector, ao_matrix, pl2);
0014 %
0015 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0016 %                   ao_in  = ao_vector
0017 %                   ao_in  = ao_matrix
0018 %
0019 % PARAMETER LIST:  <key>          <value>          <description>
0020 %                  'dim'          1 or 2 or 3 ...    takes the median along the
0021 %                                     dimension dim
0022 %                                     tsdata  fsdata  xydata
0023 %                  'xdata'        't'          'f'          'x'          compute the median of the
0024 %                                     x-axis
0025 %                  'ydata'        'x'          'xx'         'y'          compute the median of the
0026 %                                     y-axis
0027 % The following call returns a parameter list object that contains the
0028 % default parameter values:
0029 % >> pl = median(ao, 'Params')

```



```

0030 %
0031 % HISTORY: 23-05-2007 Diepholz
0032 %           Creation
0033 %
0034 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0035
0036 VERSION = '$Id: median.m,v 1.9 2007/11/26 14:48:13 ingo Exp $';
0037 ao_out   = [];
0038
0039 %% Check if this is a call for parameters
0040 if nargin == 2
0041     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0042         in = char(varargin{2});
0043         if strcmp(in, 'Params')
0044             ao_out = getDefaultPL();
0045             return
0046         elseif strcmp(in, 'Version')
0047             ao_out = VERSION;
0048             return
0049         end
0050     end
0051 end
0052
0053 %% Collect input ao's, plist's and ao variable names
0054 in_names = {};
0055 ps       = plist();
0056 for ii = 1:nargin
0057     in_names{end+1} = inputname(ii);
0058
0059     if isnumeric(varargin{ii})
0060         ps = [ps plist(param('dim', varargin{ii}))];
0061     end
0062 end
0063
0064 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0065
0066 pl = combine(pl, ps);
0067
0068 %% go through analysis objects
0069 for j=1:numel(ao_set)
0070     a = ao_set(j);
0071
0072     [h, median_data] = single_operation(a.data, 'median', pl);
0073
0074     %% Add the history from the ao object to the history
0075     h = set(h, 'inhists', [a.hist]);
0076
0077     %% Set the var_name to the history
0078     h = set(h, 'invars', cellstr(invars{j}));
0079
0080     do_xdata = find(pl, 'xdata');
0081     do_ydata = find(pl, 'ydata');

```

```

0082
0083 %% Is no axis entry in the parameter list
0084 %% then set the default axis = xdata
0085 if isempty(do_xdata) && isempty(do_ydata)
0086     do_ydata = 'yes';
0087 end
0088
0089 %% Get the operation result. It is stored in the individual axis
0090 pl_get_axis = plist([param('xdata', do_xdata) param('ydata', do_ydata)]);
0091 [median_x, median_y] = get_xy_values(median_data, pl_get_axis);
0092
0093 median_cdata = cdata([median_x median_y]);
0094 median_cdata = set (median_cdata, 'name', median_data.name);
0095 median_cdata = set (median_cdata, 'xunits', median_data.xunits);
0096 median_cdata = set (median_cdata, 'yunits', median_data.yunits);
0097
0098 %% create a new analysis objects
0099 new_ao = a;
0100 new_ao = ao (median_cdata, h);
0101 new_ao = setnh(new_ao, 'name', sprintf('median(%s)', invars{j})) );
0102
0103 ao_out = [ao_out new_ao];
0104
0105 end
0106
0107 % Reshape the ouput to the same size of the input
0108 ao_out = reshape(ao_out, size(ao_set));
0109
0110
0111 %% Get default params
0112 function pl_default = getDefaultPL()
0113
0114     pl_default = plist([param('dim',    '')
0115                          param('xdata', '')
0116                          param('ydata', '')]);
0117
0118 % END

```

[Home](#) > [classes](#) > [@ao](#) > minus.m

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 1.10 2007/11/02 13:17:41 ingo Exp \$

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

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

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

```

0001 function b = minus(varargin)
0002 % MINUS overloads - operator for analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MINUS overloads - operator for analysis objects.
0007 %
0008 % CALL:          a = a1-a2
0009 %
0010 % VERSION:       $Id: minus.m,v 1.10 2007/11/02 13:17:41 ingo Exp $
0011 %
0012 % The following call returns a parameter list object that contains the
0013 % default parameter values:
0014 %
0015 % >> pl = minus(ao, 'Params')
0016 %
0017 % HISTORY: 05-02-07 M Hewitson
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 ALGONAME = mfilename;
0023 VERSION  = '$Id: minus.m,v 1.10 2007/11/02 13:17:41 ingo Exp $';
0024
0025 %% Check if this is a call for parameters
0026 if nargin == 2
0027     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0028         in = char(varargin{2});
0029         if strcmp(in, 'Params')
0030             b = getDefaultPL();
0031             return
0032         elseif strcmp(in, 'Version')
0033             b = VERSION;
0034             return
0035         end
0036     end
0037 end
0038
0039
0040 %% capture input variable names
0041 invars = {};
0042 for j=1:nargin
0043     iname = inputname(j);
0044     if isempty(iname) && isnumeric(varargin{j})
0045         iname = num2str(varargin{j});
0046     elseif isempty(iname) && ischar(varargin{j})

```

```
0047     iname = varargin{j};
0048     end
0049     invars = [invars cellstr(iname)];
0050 end
0051
0052 op = '-';
0053
0054 [a1,a2,do] = aooperate(varargin, op);
0055
0056 %----- create output AO
0057
0058 % make a new history object
0059 h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
0060 h = set(h, 'invars', invars);
0061
0062 % get names for output
0063 if isempty(char(invars{1}))
0064     n1 = a1.name;
0065 else
0066     n1 = char(invars{1});
0067 end
0068 if isempty(char(invars{2}))
0069     n2 = a2.name;
0070 else
0071     n2 = char(invars{2});
0072 end
0073
0074
0075 % make output analysis object
0076 b = ao(do, h);
0077 b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));
0078
0079
0080
0081 %% Get default params
0082 function pl_default = getDefaultPL()
0083
0084     pl_default = plist();
0085
0086 % END
0087
```

[Home](#) > [classes](#) > [@ao](#) > mpower.m

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 1.11 2007/11/02 13:17:41 ingo Exp \$

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

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

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

```

0001 function b = mpower(varargin)
0002 % MPOWER implements power operator for analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MPOWER implements power operator for analysis objects.
0007 %               computes b^c where b is an analysis object and c is either
0008 %               another analysis object or a constant.
0009 %
0010 % CALL:         a = a1^2
0011 %               a = a1^a2      % with a2.data.vals = scalar
0012 %
0013 % VERSION:      $Id: mpower.m,v 1.11 2007/11/02 13:17:41 ingo Exp $
0014 %
0015 % The following call returns a parameter list object that contains the
0016 % default parameter values:
0017 %
0018 % >> pl = mpower(ao, 'Params')
0019 %
0020 % HISTORY: 01-02-07 M Hewitson
0021 %           Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 ALGONAME = mfilename;
0026 VERSION  = '$Id: mpower.m,v 1.11 2007/11/02 13:17:41 ingo Exp $';
0027
0028 %% Check if this is a call for parameters
0029 if nargin == 2
0030     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0031         in = char(varargin{2});
0032         if strcmp(in, 'Params')
0033             b = getDefaultPL();
0034             return
0035         elseif strcmp(in, 'Version')
0036             b = VERSION;
0037             return
0038         end
0039     end
0040 end
0041
0042 %% capture input variable names
0043 invars = {};

```

```

0044 for j=1:nargin
0045     iname = inputname(j);
0046     if isempty(iname) & isnumeric(varargin{j})
0047         iname = num2str(varargin{j});
0048     elseif isempty(iname) & ischar(varargin{j})
0049         iname = varargin{j};
0050     end
0051     invars = [invars cellstr(iname)];
0052 end
0053
0054 op = '^';
0055
0056 [a1,a2,do] = aooperate(varargin, op);
0057
0058 %----- create output AO
0059
0060 % make a new history object
0061 h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
0062 h = set(h, 'invars', invars);
0063
0064 % get names for output
0065 if isempty(char(invars{1}))
0066     n1 = a1.name;
0067 else
0068     n1 = char(invars{1});
0069 end
0070 if isempty(char(invars{2}))
0071     n2 = a2.name;
0072 else
0073     n2 = char(invars{2});
0074 end
0075
0076 % make output analysis object
0077 % op = op(2:end);
0078 b = ao(do, h);
0079 b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));
0080
0081
0082 %% Get default params
0083 function pl_default = getDefaultPL()
0084
0085     pl_default = plist();
0086
0087
0088 % END

```


[Home](#) > [classes](#) > [@ao](#) > mrdivide.m

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 1.11 2007/11/02 13:17:41 ingo Exp \$

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

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

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

```

0001 function b = mrdivide(varargin)
0002 % MRDIVIDE overloads / operator for analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MRDIVIDE overloads / operator for analysis objects.
0007 %
0008 % CALL:          a = a1/scalar
0009 %               a = a1/a2
0010 %
0011 % VERSION:       $Id: mrdivide.m,v 1.11 2007/11/02 13:17:41 ingo Exp $
0012 %
0013 % The following call returns a parameter list object that contains the
0014 % default parameter values:
0015 %
0016 % >> pl = mrdivide(ao, 'Params')
0017 %
0018 % HISTORY: 05-02-07 M Hewitson
0019 %           Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 ALGONAME = mfilename;
0024 VERSION  = '$Id: mrdivide.m,v 1.11 2007/11/02 13:17:41 ingo Exp $';
0025
0026 %% Check if this is a call for parameters
0027 if nargin == 2
0028     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0029         in = char(varargin{2});
0030         if strcmp(in, 'Params')
0031             b = getDefaultPL();
0032             return
0033         elseif strcmp(in, 'Version')
0034             b = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 %% capture input variable names
0041 invars = {};
0042 for j=1:nargin
0043     iname = inputname(j);
0044     if isempty(iname) && isnumeric(varargin{j})

```

```

0045     iname = num2str(varargin{j});
0046     elseif isempty(iname) && ischar(varargin{j})
0047         iname = varargin{j};
0048     end
0049     invars = [invars cellstr(iname)];
0050 end
0051
0052 op = '/';
0053
0054 [a1,a2,do] = aooperate(varargin, op);
0055
0056 %----- create output AO
0057
0058 % make a new history object
0059 h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
0060 h = set(h, 'invars', invars);
0061
0062 % get names for output
0063 if isempty(char(invars{1}))
0064     n1 = a1.name;
0065 else
0066     n1 = char(invars{1});
0067 end
0068 if isempty(char(invars{2}))
0069     n2 = a2.name;
0070 else
0071     n2 = char(invars{2});
0072 end
0073
0074 % make output analysis object
0075 % op = op(2:end);
0076 b = ao(do, h);
0077 b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));
0078
0079 %% Get default params
0080 function pl_default = getDefaultPL()
0081
0082     pl_default = plist();
0083
0084
0085 % END

```

[Home](#) > [classes](#) > [@ao](#) > mtimes.m

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 1.12 2007/11/02 13:17:41 ingo Exp \$

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

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

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

```

0001 function b = mtimes(varargin)
0002 % MTIMES overloads * operator for analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MTIMES overloads * operator for analysis objects.
0007 %               This just does the same as times (*) for analysis objects.
0008 %
0009 % CALL:         a = a1*scalar
0010 %               a = a1*a2
0011 %
0012 % VERSION:      $Id: mtimes.m,v 1.12 2007/11/02 13:17:41 ingo Exp $
0013 %
0014 % The following call returns a parameter list object that contains the
0015 % default parameter values:
0016 %
0017 % >> pl = mtimes(ao, 'Params')
0018 %
0019 % HISTORY: 05-02-07 M Hewitson
0020 %           Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 ALGONAME = mfilename;
0025 VERSION  = '$Id: mtimes.m,v 1.12 2007/11/02 13:17:41 ingo Exp $';
0026
0027 %% Check if this is a call for parameters
0028 if nargin == 2
0029     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0030         in = char(varargin{2});
0031         if strcmp(in, 'Params')
0032             b = getDefaultPL();
0033             return
0034         elseif strcmp(in, 'Version')
0035             b = VERSION;
0036             return
0037         end
0038     end
0039 end
0040
0041 % capture input variable names
0042 invars = {};
0043 for j=1:nargin
0044     iname = inputname(j);

```

```

0045     if isempty(iname) && isnumeric(varargin{j})
0046         iname = num2str(varargin{j});
0047     elseif isempty(iname) && ischar(varargin{j})
0048         iname = varargin{j};
0049     end
0050     invars = [invars cellstr(iname)];
0051 end
0052
0053 op = '*';
0054
0055 [a1,a2,do] = aooperate(varargin, op);
0056
0057 %----- create output AO
0058
0059 % make a new history object
0060 h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
0061 h = set(h, 'invars', invars);
0062
0063 % get names for output
0064 if isempty(char(invars{1}))
0065     n1 = a1.name;
0066 else
0067     n1 = char(invars{1});
0068 end
0069 if isempty(char(invars{2}))
0070     n2 = a2.name;
0071 else
0072     n2 = char(invars{2});
0073 end
0074
0075 % make output analysis object
0076 % op = op(2:end);
0077 b = ao(do, h);
0078 b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));
0079
0080 %% Get default params
0081 function pl_default = getDefaultPL()
0082
0083     pl_default = plist();
0084
0085
0086 % END

```

norm

PURPOSE

NORM overloads the norm operator for Analysis Objects.

SYNOPSIS

function varargout = norm(varargin)

DESCRIPTION

```
NORM overloads the norm operator for Analysis Objects.

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

DESCRIPTION:NORM overloads the norm operator for Analysis Objects.

CALL:      b = norm (a,pl) % only with data = cdata
           b = norm (a)

INPUTS:    pl    - a parameter list
           a      - input analysis object

OUTPUTS:    like matlab fct

Parameters: 'option' - a string or value that can be submitted i.e. 'inf' for
                    the infinity norm of the input ao, the largest row sum
                    options are the same as for the matlab function

VERSION:    $Id: norm.m,v 1.15 2008/01/20 12:53:29 hewitson Exp $

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

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

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

```

0001 function varargout = norm(varargin)
0002 % NORM overloads the norm operator for Analysis Objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION:NORM overloads the norm operator for Analysis Objects.
0007 %
0008 % CALL:          b = norm (a,pl) % only with data = cdata
0009 %               b = norm (a)
0010 %
0011 % INPUTS:       pl   - a parameter list
0012 %               a    - input analysis object
0013 %
0014 % OUTPUTS:      like matlab fct
0015 %
0016 % Parameters: 'option' - a string or value that can be submitted i.e. 'inf' for
0017 %               the infinity norm of the input ao, the largest row sum
0018 %               options are the same as for the matlab function
0019 %
0020 % VERSION:      $Id: norm.m,v 1.15 2008/01/20 12:53:29 hewitson Exp $
0021 %
0022 % The following call returns a parameter list object that contains the
0023 % default parameter values:
0024 %
0025 % >> pl = norm(ao, 'Params')
0026 %
0027 % HISTORY: 08-05-07 A Monsky
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 %% Check if this is a call for parameters
0033
0034 VERSION = '$Id: norm.m,v 1.15 2008/01/20 12:53:29 hewitson Exp $';
0035 bs      = [];

```



```

0036
0037 if nargin == 2
0038     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0039         in = char(varargin{2});
0040         if strcmp(in, 'Params')
0041             varargout{1} = getDefaultPL();
0042             return
0043         elseif strcmp(in, 'Version')
0044             varargout{1} = VERSION;
0045             return
0046         end
0047     end
0048 end
0049
0050 %% Collect input ao's, plist's and ao variable names
0051 in_names = {};
0052 for ii = 1:nargin
0053     in_names{end+1} = inputname(ii);
0054 end
0055
0056 [as, pl, invars] = collect_inputs(varargin, in_names);
0057
0058 if ~isempty(pl)
0059     pl = combine(pl);
0060 end
0061
0062
0063 %% go through analysis objects
0064 for j=1:numel(as)
0065     a = as(j);
0066
0067     d = get(a, 'data');
0068     dinfo = whos('d');
0069     % Which data type do we have
0070     dtype = dinfo.class;
0071     switch dtype
0072     case 'cdata'
0073         %disp('* NORM of cdata object');
0074         % make a new cdata object
0075         [h, c] = single_operation(d, 'norm', pl);
0076
0077         % make output analysis object
0078         h = set(h, 'inhists', [a.hist]);
0079
0080         %% Set the var_name to the history
0081         h = set(h, 'invars', cellstr(invars{j}));
0082
0083         b = ao(c, h);
0084         b = setnh(b, 'name', sprintf('norm(%s)', char(invars{j})));
0085     case {'tsdata', 'fsdata', 'xydata'}
0086         error('### this function works for cdata type AO only')

```

```
0087     otherwise
0088         error('### unknown data type.')
0089
0090     end
0091
0092     % add to output
0093     bs = [bs b];
0094 end
0095
0096 % Reshape the ouput to the same size of the input
0097 bs = reshape(bs, size(as));
0098 varargout{1} = bs;
0099
0100 %% Get default params
0101 function plo = getDefaultPL()
0102
0103 plo = plist();
0104 plo = append(plo, 'option', []);
0105
0106
0107 % END
```

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 1.5 2008/01/20 12:55:04 hewitson 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')

%%%

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:

- **upsample** UPSAMPLE overloads upsample function for AOs.

SUBFUNCTIONS

- **function pl_default = getDefaultPL()**

SOURCE CODE

```

0001 function ao_out = phase(varargin)
0002 % PHASE overloads the ltpda_phase operator for Analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: PHASE overloads the ltpda_phase operator for Analysis objects.
0007 %
0008 % CALL:          ao_out = phase(ao_in);
0009 %               ao_out = phase(ao_in, pl);
0010 %
0011 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0012 %                  ao_in  = ao_vector
0013 %                  ao_in  = ao_matrix
0014 %
0015 % PARAMETER LIST:  <key>          <value>          <description>
0016 %                  tsdata  fsdata  xydata
0017 %                  'xdata'  't'      'f'      'x'      compute the xdata
0018 %                  'ydata'  'x'      'xx'     'y'      compute the ydata
0019 %
0020 % VERSION:         $Id: phase.m,v 1.5 2008/01/20 12:55:04 hewitson Exp $
0021 %
0022 % HISTORY:         22-08-2007 M Hewitson
0023 %                  Creation
0024 %
0025 % The following call returns a parameter list object that contains the
0026 % default parameter values:
0027 %
0028 % >> pl = phase(ao, 'Params')
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION  = '$Id: phase.m,v 1.5 2008/01/20 12:55:04 hewitson Exp $';
0033 ao_out   = [];
0034

```

```

0035 %% Check if this is a call for parameters
0036 if nargin == 2
0037     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0038         in = char(varargin{2});
0039         if strcmp(in, 'Params')
0040             ao_out = getDefaultPL();
0041             return
0042         elseif strcmp(in, 'Version')
0043             ao_out = VERSION;
0044             return
0045         end
0046     end
0047 end
0048
0049 %% Collect input ao's, plist's and ao variable names
0050 in_names = {};
0051 for ii = 1:nargin
0052     in_names{end+1} = inputname(ii);
0053 end
0054
0055 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0056
0057 %% go through analysis objects
0058 for j=1:numel(ao_set)
0059     a = ao_set(j);
0060
0061     [h, phase_data] = single_operation(a.data, 'phase', pl);
0062
0063     %% Set the x, y-units to their old values (without phase())
0064     phase_data = set(phase_data, 'xunits', a.data.xunits);
0065     phase_data = set(phase_data, 'yunits', a.data.yunits);
0066     phase_data = set(phase_data, 'name', sprintf('phase(%s)', a.data.name));
0067
0068     %% Add the history from the ao object to the history
0069     h = set(h, 'inhists', [a.hist]);
0070
0071     %% Set the var_name to the history
0072     h = set(h, 'invars', cellstr(invars{j}));
0073
0074     %% create a new analysis objects
0075     new_ao = ao (phase_data, h);
0076     new_ao = setnh(new_ao, 'name', sprintf('phase(%s)', invars{j}) );
0077
0078     ao_out = [ao_out new_ao];
0079
0080 end
0081
0082 % Reshape the output to the same size of the input
0083 ao_out = reshape(ao_out, size(ao_set));
0084
0085

```

```
0086 %% Get default params
0087 function pl_default = getDefaultPL()
0088
0089     pl_default = plist([param('xdata', '')
0090                          param('ydata', '')] );
0091
0092 % END
```

[Home](#) > [classes](#) > [@ao](#) > plus.m

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 1.14 2007/11/02 13:17:41 ingo Exp \$

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

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

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

```

0001 function b = plus(varargin)
0002 % PLUS implements addition operator for two analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: PLUS implements addition operator for two analysis objects.
0007 %
0008 % CALL:          a = a1+scalar
0009 %               a = a1+a2
0010 %
0011 % VERSION:       $Id: plus.m,v 1.14 2007/11/02 13:17:41 ingo Exp $
0012 %
0013 % The following call returns a parameter list object that contains the
0014 % default parameter values:
0015 %
0016 % >> pl = plus(ao, 'Params')
0017 %
0018 % HISTORY: 01-02-07 M Hewitson
0019 %           Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 VERSION = '$Id: plus.m,v 1.14 2007/11/02 13:17:41 ingo Exp $';
0024
0025 %% Check if this is a call for parameters
0026 if nargin == 2
0027     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0028         in = char(varargin{2});
0029         if strcmp(in, 'Params')
0030             b = getDefaultPL();
0031             return
0032         elseif strcmp(in, 'Version')
0033             b = VERSION;
0034             return
0035         end
0036     end
0037 end
0038
0039 %% capture input variable names
0040 invars = {};
0041 for j=1:nargin
0042     iname = inputname(j);
0043     if isempty(iname) && isnumeric(varargin{j})
0044         iname = num2str(varargin{j});

```



```

0045     elseif isempty(iname) && ischar(varargin{j})
0046         iname = varargin{j};
0047     end
0048     invars = [invars cellstr(iname)];
0049 end
0050
0051 ALGONAME = mfilename;
0052 VERSION  = '$Id: plus.m,v 1.14 2007/11/02 13:17:41 ingo Exp $';
0053
0054 op = '+';
0055
0056 [a1,a2,do] = aooperate(varargin, op);
0057
0058 %----- create output AO
0059
0060 % make a new history object
0061 h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
0062 h = set(h, 'invars', invars);
0063
0064 % get names for output
0065 if isempty(char(invars{1}))
0066     n1 = a1.name;
0067 else
0068     n1 = char(invars{1});
0069 end
0070 if isempty(char(invars{2}))
0071     n2 = a2.name;
0072 else
0073     n2 = char(invars{2});
0074 end
0075
0076
0077 % make output analysis object
0078 b = ao(do, h);
0079 b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));
0080
0081
0082 %% Get default params
0083 function pl_default = getDefaultPL()
0084
0085     pl_default = plist();
0086
0087 % END

```

[Home](#) > [classes](#) > [@ao](#) > polyfit.m

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 1.10 2007/11/26 14:48:13 ingo 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')
```

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

```

0001 function varargout = polyfit(varargin)
0002 % POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: POLYFIT overloads polyfit() function of MATLAB for Analysis
0007 %               Objects.
0008 %
0009 % CALL:         b = polyfit(a, pl)
0010 %
0011 % VERSION:      $Id: polyfit.m,v 1.10 2007/11/26 14:48:13 ingo Exp $
0012 %
0013 % Parameters:   'N'          - degree of polynomial to fit
0014 %               'coeffs'     - (optional) coefficients
0015 %                               formed e.g. by [p,s] = polyfit(x,y,N);
0016 %
0017 % The following call returns a parameter list object that contains the
0018 % default parameter values:
0019 %
0020 % >> pl = polyfit(ao, 'Params')
0021 %
0022 % HISTORY: 05-06-07 M Hewitson
0023 %           Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026
0027 ALGONAME = mfilename;
0028 VERSION  = '$Id: polyfit.m,v 1.10 2007/11/26 14:48:13 ingo Exp $';
0029 bs       = [];
0030
0031 %% Check if this is a call for parameters
0032 if nargin == 2
0033     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0034         in = char(varargin{2});
0035         if strcmp(in, 'Params')
0036             varargout{1} = getDefaultPL();
0037         return

```

```

0038     elseif strcmp(in, 'Version')
0039         varargout{1} = VERSION;
0040         return
0041     end
0042 end
0043 end
0044
0045 %% Collect input ao's, plist's and ao variable names
0046 in_names = {};
0047 for ii = 1:nargin
0048     in_names{end+1} = inputname(ii);
0049 end
0050
0051 [as, ps, invars] = collect_inputs(varargin, in_names);
0052
0053 % check plist
0054 if isempty(ps)
0055     pl = getDefaultPL();
0056 else
0057     pl = combine(ps, getDefaultPL);
0058 end
0059
0060
0061 % Degree of polynomial to fit
0062 N      = find(pl, 'N');
0063 coeffs = find(pl, 'coeffs');
0064
0065 %-----
0066 % Loop over input AOs
0067 for j=1:numel(as)
0068     a = as(j);
0069
0070     %-----
0071     % Get data
0072     [x,y] = get_xy_values(a.data);
0073
0074     %-----
0075     % Fit polynomial
0076
0077     if isempty(coeffs)
0078         %[p,s,mu] = polyfit(x,y,N);
0079         [p,s] = polyfit(x,y,N);
0080
0081         % Add coefficients to plist
0082         pl = pset(pl, 'coeffs', p);
0083     else
0084         p = coeffs;
0085     end
0086
0087     % make new values from these coefficients
0088     y = polyval(p, x);
0089

```

```

0090
0091  %-----
0092  % Make output A0/cdata
0093  d = a.data;
0094  d = set_xy_axis(d, x, y);
0095  d = set(d, 'name', sprintf('polyfit(%s)', d.name));
0096
0097  h = history(ALGONAME, VERSION, pl, a.hist);
0098  h = set(h, 'invars', invars(j));
0099
0100  % make output analysis object
0101  b = ao(d, h);
0102
0103  % set name
0104  b = setnh(b, 'name', sprintf('polyfit(%s)', invars{j}));
0105
0106  % add to outputs
0107  bs = [bs b];
0108
0109 end
0110
0111 % Reshape the ouput to the same size of the input
0112 bs = reshape(bs, size(as));
0113
0114 %-----
0115 % Set outputs
0116 varargout{1} = bs;
0117
0118 %-----
0119 % Get default params
0120 function plo = getDefaultPL()
0121
0122 plo = plist();
0123 plo = append(plo, param('N', 1));
0124 plo = append(plo, param('coeffs', []));
0125
0126 % END
0127

```

[Home](#) > [classes](#) > [@ao](#) > power.m

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: $a = a1.^{\text{scalar}}$

VERSION: \$Id: power.m,v 1.9 2007/11/02 13:17:41 ingo Exp \$

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

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

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

```

0001 function b = power(varargin)
0002 % POWER implements power operator for analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: POWER implements power operator for analysis objects.
0007 %               computes b.^c where b is an analysis object and c is
0008 %               either another analysis object or a constant.
0009 %
0010 % CALL:         a = a1.^scalar
0011 %
0012 % VERSION:      $Id: power.m,v 1.9 2007/11/02 13:17:41 ingo Exp $
0013 %
0014 % The following call returns a parameter list object that contains the
0015 % default parameter values:
0016 %
0017 % >> pl = power(ao, 'Params')
0018 %
0019 % HISTORY: 01-02-07 M Hewitson
0020 %           Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 ALGONAME = mfilename;
0025 VERSION  = '$Id: power.m,v 1.9 2007/11/02 13:17:41 ingo Exp $';
0026
0027 %% Check if this is a call for parameters
0028 if nargin == 2
0029     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0030         in = char(varargin{2});
0031         if strcmp(in, 'Params')
0032             b = getDefaultPL();
0033             return
0034         elseif strcmp(in, 'Version')
0035             b = VERSION;
0036             return
0037         end
0038     end
0039 end
0040
0041 %% capture input variable names
0042 invars = {};
0043 for j=1:nargin
0044     iname = inputname(j);

```

```

0045     if isempty(iname) & isnumeric(varargin{j})
0046         iname = num2str(varargin{j});
0047     elseif isempty(iname) & ischar(varargin{j})
0048         iname = varargin{j};
0049     end
0050     invars = [invars cellstr(iname)];
0051 end
0052
0053 op = ' .^ ';
0054
0055 [a1,a2,do] = aooperate(varargin, op);
0056
0057 %----- create output AO
0058
0059 % make a new history object
0060 h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
0061 h = set(h, 'invars', invars);
0062
0063 % get names for output
0064 if isempty(char(invvars{1}))
0065     n1 = a1.name;
0066 else
0067     n1 = char(invvars{1});
0068 end
0069 if isempty(char(invvars{2}))
0070     n2 = a2.name;
0071 else
0072     n2 = char(invvars{2});
0073 end
0074
0075 % make output analysis object
0076 % op = op(2:end);
0077 b = ao(do, h);
0078 b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));
0079
0080 %% Get default params
0081 function pl_default = getDefaultPL()
0082
0083     pl_default = plist();
0084
0085
0086 % END
0087

```


[Home](#) > [classes](#) > [@ao](#) > rdivide.m

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 1.11 2007/11/02 13:17:41 ingo Exp \$

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

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

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

```

0001 function b = rdivide(varargin)
0002 % RDIVIDE overloads ./ operator for analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: RDIVIDE overloads ./ operator for analysis objects.
0007 %
0008 % CALL:          a = a1./scalar
0009 %
0010 % VERSION:       $Id: rdivide.m,v 1.11 2007/11/02 13:17:41 ingo Exp $
0011 %
0012 % The following call returns a parameter list object that contains the
0013 % default parameter values:
0014 %
0015 % >> pl = rdivide(ao, 'Params')
0016 %
0017 % HISTORY: 05-02-07 M Hewitson
0018 %          Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 ALGONAME = mfilename;
0023 VERSION  = '$Id: rdivide.m,v 1.11 2007/11/02 13:17:41 ingo Exp $';
0024
0025 %% Check if this is a call for parameters
0026 if nargin == 2
0027     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0028         in = char(varargin{2});
0029         if strcmp(in, 'Params')
0030             b = getDefaultPL();
0031             return
0032         elseif strcmp(in, 'Version')
0033             b = VERSION;
0034             return
0035         end
0036     end
0037 end
0038
0039 %% capture input variable names
0040 invars = {};
0041 for j=1:nargin
0042     iname = inputname(j);
0043     if isempty(iname) && isnumeric(varargin{j})
0044         iname = num2str(varargin{j});
0045     elseif isempty(iname) && ischar(varargin{j})
0046         iname = varargin{j};

```

```

0047     end
0048     invars = [invars cellstr(iname)];
0049 end
0050
0051 op = './';
0052
0053 [a1,a2,do] = aooperate(varargin, op);
0054
0055 %----- create output AO
0056
0057 % make a new history object
0058 h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
0059 h = set(h, 'invars', invars);
0060
0061 % get names for output
0062 if isempty(char(invars{1}))
0063     n1 = a1.name;
0064 else
0065     n1 = char(invars{1});
0066 end
0067 if isempty(char(invars{2}))
0068     n2 = a2.name;
0069 else
0070     n2 = char(invars{2});
0071 end
0072
0073 % make output analysis object
0074 % op = op(2:end);
0075 b = ao(do, h);
0076 b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));
0077
0078 %% Get default params
0079 function pl_default = getDefaultPL()
0080
0081     pl_default = plist();
0082
0083 % END

```

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.

%%%

DESCRIPTION: REAL overloads the real operator for Analysis objects.

CALL: ao_out = real(ao_in);
 ao_out = real(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 = real(ao, 'Params')

VERSION: \$Id: real.m,v 1.4 2007/11/26 14:48:13 ingo Exp \$

HISTORY: 20-0802007 Diepholz
 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:

- **export** EXPORT export an analysis object to a text file.
- **ipLOT** IPLOT provides an intelligent plotting tool for LTPDA.
- **plot** PLOT a simple plot of analysis objects.
- **plot_old** PLOT plots an array of analysis objects.
- **simple_plot** PLOT a simple plot of analysis objects.

SUBFUNCTIONS

- **function pl_default = getDefaultPL()**

SOURCE CODE

```

0001 function ao_out = real(varargin)
0002 % REAL overloads the real operator for Analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: REAL overloads the real operator for Analysis objects.
0007 %
0008 % CALL: ao_out = real(ao_in);
0009 %         ao_out = real(ao_in, pl);
0010 %
0011 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0012 %                   ao_in  = ao_vector
0013 %                   ao_in  = ao_matrix
0014 %
0015 % PARAMETER LIST:  <key>          <value>          <description>
0016 %                   tsdata    fsdata    xydata
0017 %                   'xdata'    't'       'f'       'x'       compute the xdata
0018 %                   'ydata'    'x'       'xx'      'y'       compute the ydata
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = real(ao, 'Params')
0024 %
0025 % VERSION: $Id: real.m,v 1.4 2007/11/26 14:48:13 ingo Exp $
0026 %
0027 % HISTORY: 20-0802007 Diepholz
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

0031
0032 VERSION = '$Id: real.m,v 1.4 2007/11/26 14:48:13 ingo Exp $';
0033 ao_out = [];
0034
0035 %% Check if this is a call for parameters
0036 if nargin == 2
0037     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0038         in = char(varargin{2});
0039         if strcmp(in, 'Params')
0040             ao_out = getDefaultPL();
0041             return
0042         elseif strcmp(in, 'Version')
0043             ao_out = VERSION;
0044             return
0045         end
0046     end
0047 end
0048
0049 %% Collect input ao's, plist's and ao variable names
0050 in_names = {};
0051 for ii = 1:nargin
0052     in_names{end+1} = inputname(ii);
0053 end
0054
0055 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0056
0057 if ~isempty(pl)
0058     pl = combine(pl);
0059 end
0060
0061 %% go through analysis objects
0062 for j=1:numel(ao_set)
0063     a = ao_set(j);
0064
0065     [h, real_data] = single_operation(a.data, 'real', pl);
0066
0067     %% Add the history from the ao object to the history
0068     h = set(h, 'inhists', [a.hist]);
0069
0070     %% Set the var_name to the history
0071     h = set(h, 'invars', cellstr(invars{j}));
0072
0073     %% create a new analysis objects
0074     new_ao = ao (real_data, h);
0075     new_ao = setnh(new_ao, 'name', sprintf('real(%s)', invars{j}) );
0076
0077     ao_out = [ao_out new_ao];
0078
0079 end
0080
0081 % Reshape the ouput to the same size of the input

```

```
0082 ao_out = reshape(ao_out, size(ao_set));
0083
0084
0085 %% Get default params
0086 function pl_default = getDefaultPL()
0087
0088     pl_default = plist([param('xdata', '')
0089                          param('ydata', '')]);
0090
0091
0092 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > resample.m

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 1.13 2007/11/26 14:48:13 ingo Exp \$

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

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

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

```

0001 function bs = resample(varargin)
0002 % RESAMPLE overloads resample function for AOs.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: RESAMPLE overloads resample function for AOs.
0007 %
0008 % CALL:          b = resample(a, pl)
0009 %
0010 % Parameters: 'fsout' - specify the desired output sample rate
0011 %
0012 % VERSION:       $Id: resample.m,v 1.13 2007/11/26 14:48:13 ingo Exp $
0013 %
0014 % The following call returns a parameter list object that contains the
0015 % default parameter values:
0016 %
0017 % >> pl = resample(ao, 'Params')
0018 %
0019 % HISTORY: 19-03-07 M Hewitson
0020 %          Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: resample.m,v 1.13 2007/11/26 14:48:13 ingo Exp $';
0025 bs      = [];
0026
0027 %% Check if this is a call for parameters
0028 if nargin == 2
0029     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0030         in = char(varargin{2});
0031         if strcmp(in, 'Params')
0032             bs = getDefaultPL();
0033             return
0034         elseif strcmp(in, 'Version')
0035             bs = VERSION;
0036             return
0037         end
0038     end
0039 end
0040

```

```

0041 ALGNAME = mfilename;
0042 VERSION  = '$Id: resample.m,v 1.13 2007/11/26 14:48:13 ingo Exp $';
0043
0044 %% capture input variables
0045 invars = {};
0046 as      = [];
0047 pl      = [];
0048
0049 for j=1:nargin
0050     a = varargin{j};
0051     if isa(a, 'ao')
0052         d = a.data;
0053         if isa(d, 'tsdata')
0054             as = [as a];
0055
0056             ao_name = inputname(j);
0057             if isempty(ao_name)
0058                 ao_name = 'no_ao_name';
0059             end
0060
0061             % Memorise the variable name of the corresponding analysis object.
0062             % If the ao is an array or vector add the index to the variable name
0063             if numel(varargin{j}) == 1
0064                 invars{end+1} = ao_name;
0065             else
0066                 for ii=1:numel(varargin{j})
0067                     [I,J] = ind2sub(size(varargin{j}),ii);
0068                     invars{end+1} = sprintf('%s(%d,%d)', ao_name, I, J);
0069                 end
0070             end
0071         else
0072             warning('!!! ignoring non-time-series AO.');
```

```

0093
0094 for j=1:numel(as)
0095
0096     d = as(j).data;
0097
0098     % resample y
0099     if isempty(filt)
0100         [y,b] = resample(d.y, fsout, d.fs);
0101     else
0102         [p,q] = rat( fsout/d.fs, 1e-12 );
0103         b = p*filt.a;
0104         y = resample(d.y, fsout, d.fs, b);
0105     end
0106
0107
0108     % make new output tsdata
0109     do = tsdata(y, fsout);
0110     d = set(d, 'x', do.x);
0111     d = set(d, 'y', do.y);
0112     d = set(d, 'fs', do.fs);
0113
0114     %----- create output A0
0115
0116     % make a new history object
0117     h = history(ALGONAME, VERSION, pl, as(j).hist);
0118     h = set(h, 'invars', invars);
0119
0120     % make output analysis object
0121     b = ao(d, h);
0122     b = setnh(b, 'name', sprintf('resample(%s)', invars{j}));
0123
0124     bs = [bs b];
0125
0126 end
0127
0128 % Reshape the ouput to the same size of the input
0129 bs = reshape(bs, size(as));
0130
0131 % Get default params
0132 function plo = getDefaultPL()
0133
0134 plo = plist();
0135 plo = append(plo, param('fsout', 100));
0136
0137 % END

```

[Home](#) > [classes](#) > [@ao](#) > select.m

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 1.15 2007/11/26 14:48:13 ingo Exp \$

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

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

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

```

0001 function bo = select(varargin)
0002 % SELECT select particular samples from the input AOs and return new AOs with
0003 % only those samples.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SELECT select particular samples from the input AOs and return
0007 % new AOs with only those samples.
0008 %
0009 % CALL:      b = select(a, [1 2 3 4]);
0010 %           b = select(a, pl)
0011 %
0012 % Parameters: 'samples' - a list of samples to select
0013 %
0014 % VERSION:    $Id: select.m,v 1.15 2007/11/26 14:48:13 ingo Exp $
0015 %
0016 % The following call returns a parameter list object that contains the
0017 % default parameter values:
0018 %
0019 % >> pl = select(ao, 'Params')
0020 %
0021 % HISTORY: 14-05-07 M Hewitson
0022 %           Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 ALGONAME = mfilename;
0027 VERSION  = '$Id: select.m,v 1.15 2007/11/26 14:48:13 ingo Exp $';
0028 samples  = [];
0029
0030 %% Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             bo = getDefaultPL();
0036             return
0037         elseif strcmp(in, 'Version')
0038             bo = VERSION;
0039             return

```

```

0040     end
0041 end
0042 end
0043
0044 %% Collect input ao's, plist's and ao variable names
0045 in_names = {};
0046 for ii = 1:nargin
0047     in_names{end+1} = inputname(ii);
0048
0049     if isnumeric(varargin{ii})
0050         samples = [samples varargin{ii}];
0051     end
0052 end
0053
0054 [as, ps, invars] = collect_inputs(varargin, in_names);
0055
0056 if isempty(as)
0057     error('### Please input at least one A0.');
```

```

0058 end
0059 if isempty(ps) && isempty(samples)
0060     error('### Please specify samples to select directly or as a plist.');
```

```

0061 end
0062
0063 % Combine plists
0064 if ~isempty(ps)
0065     pl = combine(ps);
0066 else
0067     pl = plist();
0068 end
0069
0070 % Get sample selection from plist
0071 samples = [samples find(pl, 'samples')];
0072
0073 % Loop over input AOs
0074 bo = [];
0075 for j=1:numel(as)
0076
0077     a = as(j);
0078     d = a.data;
0079
0080     [x,y] = get_xy_values(d);
0081
0082     if isa(d, 'cdata')
0083         if length(d.x) == length(d.y)
0084             d = set_xy_axis(d, x(samples), y(samples));
0085         else
0086             d = set_xy_axis(d, x, y(samples));
0087         end
0088     else
0089         d = set_xy_axis(d, x(samples), y(samples));
0090     end
0091

```

```
0092 % Make output A0
0093
0094 % create new output history
0095 h = history(ALGONAME, VERSION, plist(param('samples', samples)), a.hist);
0096 h = set(h, 'invars', invars(j));
0097
0098 % make output analysis object
0099 b = ao(d, h);
0100
0101 % set name
0102 % name for this object
0103 b = setnh(b, 'name', sprintf('select(%s)', invars{j}));
0104
0105 % Add to output array
0106 bo = [bo b];
0107
0108 end
0109
0110 % Reshape the output to the same size of the input
0111 bo = reshape(bo, size(as));
0112
0113 % Get default params
0114 function plo = getDefaultPL()
0115
0116 plo = plist();
0117 plo = append(plo, param('samples', []));
0118
0119 % END
```

[Home](#) > [classes](#) > [@ao](#) > setnh.m

setnh

PURPOSE

SET set an analysis object property without recording the history.

SYNOPSIS

function ao = setnh(ao, 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 pairs
      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.yunits = 'yunits';
```

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

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

VERSION: \$Id: setnh.m,v 1.3 2007/11/14 16:30:18 ingo Exp \$

HISTORY: 30-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [ao](#) AO analysis object class constructor.
- [char](#) CHAR overloads char() function for analysis objects.
- [set](#) SET set an analysis object property.
- [setnh](#) SET set an analysis object property without recording the history.

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.
- [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.
- [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.
- [decimate](#) DECIMATE AOs containing time-series data.
- [det](#) DET overloads the determinant function for Analysis objects.
- [diag](#) DIAG overloads the diagonal operator for Analysis Objects.
- [eig](#) EIG overloads the determinant function for Analysis objects.
- [exp](#) EXP overloads the exp operator for Analysis objects. Exponential.
- [fft](#) FFT overloads the fft operator for Analysis objects.
- [filter](#) FILTER overrides the filter function for analysis objects.
- [filtfilt](#) FILTFILT overrides the filtfilt function for analysis objects.
- [find](#) FIND particular samples that satisfy the input query and return a new AO.
- [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.
- [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.
- **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.
- **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.

SUBFUNCTIONS

- **function plo = getDefaultPL()**

SOURCE CODE

```

0001 function ao = setnh(ao, varargin)
0002 % SET set an analysis object property without recording the history.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET set an analysis object property without recording the
history.
0007 %
0008 % CALL: ao = setnh(ao, 'name',      'name');
0009 %      ao = setnh(ao, 'provenance', provenance-object);
0010 %      ao = setnh(ao, 'description','description');
0011 %      ao = setnh(ao, 'hist',      history-object);
0012 %      ao = setnh(ao, 'data',      data);
0013 %      ao = setnh(ao, 'mfile',     mfile);
0014 %      ao = setnh(ao, 'mfilename', 'mfilename');
0015 %      ao = setnh(ao, 'mdlfile',   'mdlfile');
0016 %      ao = setnh(ao, 'mdlfilename','mdlfilename');
0017 %      ao = setnh(ao, 'version',   'cvs-version');
0018 %      ao = setnh(ao, 'created',   time-object);
0019 %      ao = setnh(ao,  plist-object);      --> plist-object with key/value
pairs
0020 %      ao = setnh(ao, 'fs',        fs);      --> ao.data.fs = fs;
0021 %      ao = setnh(ao, 't0',        t0);      --> ao.data.t0 = t0;
0022 %      ao = setnh(ao, 'xunits',    'xunits'); --> ao.data.xunits = 'xunits';

```

```

0023 %         ao = setnh(ao, 'yunits',      'yunits'); --> ao.data.xunits = 'yunits';
0024 %
0025 % The following call returns a parameter list object that contains the
0026 % default parameter values:
0027 %
0028 % >> pl = ao(ao, 'Params')
0029 %
0030 % VERSION: $Id: setnh.m,v 1.3 2007/11/14 16:30:18 ingo Exp $
0031 %
0032 % HISTORY: 30-01-07 M Hewitson
0033 %           Creation
0034 %
0035 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0036
0037 VERSION = '$Id: setnh.m,v 1.3 2007/11/14 16:30:18 ingo Exp $';
0038
0039 % Check if this is a call for parameters or for the cvs-version number
0040 if nargin == 2
0041     if isa(ao, 'ao') && ischar(varargin{1})
0042         in = char(varargin{1});
0043         if strcmp(in, 'Params')
0044             ao = getDefaultPL();
0045             return
0046         elseif strcmp(in, 'Version')
0047             ao = VERSION;
0048             return
0049         end
0050     end
0051 end
0052
0053 if (nargout ~= 0)
0054
0055 %     hist_pl    = plist();
0056     fields      = fieldnames(ao);
0057     propArgin = varargin;
0058
0059     if nargin >= 2 && isa(propArgin{1}, 'plist')
0060         pl      = propArgin{1};
0061         propArgin = {};
0062         for ii = 1:nparams(pl)
0063             propArgin{end+1} = pl.params(ii).key;
0064             propArgin{end+1} = pl.params(ii).val;
0065         end
0066     end
0067
0068
0069     while length(propArgin) >= 2
0070         prop = propArgin{1};
0071         val  = propArgin{2};
0072         propArgin = propArgin(3:end);
0073
0074         if ~ismember(prop, [fields; 't0'; 'fs'; 'xunits'; 'yunits'])

```

```

0075     error(['### ', prop, ' is not a valid analysis object property.']);
0076 else
0077
0078     %% For all analysis objects (ao can be a vector or a matrix)
0079     for jj = 1:numel(ao)
0080
0081         %% Also pass through some parameters to data object
0082         if strcmp(prop, 'fs')
0083             d = ao(jj).data;
0084             if isa(d, 'tsdata') || isa(d, 'fsdata')
0085                 d = setnh(d, 'fs', val);
0086             else
0087                 error('### No ''fs'' property for this data type');
0088             end
0089             ao(jj).data = d;
0090
0091         elseif strcmp(prop, 't0')
0092
0093             d = ao(jj).data;
0094             if isa(d, 'tsdata')
0095                 d = setnh(d, 't0', val);
0096             else
0097                 error('### The data object is not from the class tsdata');
0098             end
0099             ao(jj).data = d;
0100
0101         elseif strcmp(prop, 'xunits')
0102             d = ao(jj).data;
0103             d = set(d, 'xunits', val);
0104             ao(jj).data = d;
0105
0106         elseif strcmp(prop, 'yunits')
0107             d = ao(jj).data;
0108             d = set(d, 'yunits', val);
0109             ao(jj).data = d;
0110
0111         else
0112             ao(jj).(prop) = val;
0113         end
0114
0115     end % for
0116
0117 end
0118
0119 %     hist_pl = append(hist_pl, prop, val);
0120
0121 end
0122
0123 %     for jj = 1:numel(ao)
0124 %         h = history('setnh', VERSION, hist_pl);

```

```

0125 %         h = setnh(h, 'inhists', ao(jj).hist);
0126 %         h = setnh(h, 'invars', cellstr(inputname(1)));
0127 %
0128 %         ao(jj).hist = h;
0129 %     end
0130
0131 else
0132     if ischar(varargin{2})
0133         error('### please use: %s = setnh(%s, '%s'', '%s'');', ...
0134             inputname(1), ...
0135             inputname(1), ...
0136             varargin{1}, ...
0137             varargin{2});
0138     elseif isnumeric(varargin{2})
0139         error('### please use: %s = setnh(%s, '%s'', %d);', ...
0140             inputname(1), ...
0141             inputname(1), ...
0142             varargin{1}, ...
0143             varargin{2});
0144     else
0145         error('### please use: %s = setnh(%s, 'key'', 'value'');', ...
0146             inputname(1), ...
0147             inputname(1));
0148     end
0149
0150 end
0151
0152
0153 % Get default params
0154 function plo = getDefaultPL()
0155
0156 plo = plist();

```

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')

VERSION: \$Id: sin.m,v 1.17 2007/11/26 14:48:13 ingo Exp \$

HISTORY: 07-05-2007 Diepholz
Creation

SEE ALSO: ao/cos, ao/tan, ao/asin, [acos](#), [atan](#), [Contents](#)

SEE: [HELP: All Analysis Object Functions](#)

[DOC: All Analysis Object Functions:](matlab: helpwin ao/Contents;)

%%%

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

```

0001 function ao_out = sin (varargin)
0002 % SIN overloads the sin operator for Analysis objects. Sine of argument in
0003 % radians.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SIN overloads the sin operator for Analysis objects.
0007 %               Sine of argument in radians.
0008 %               SIN(ao) is the sine of the elements of ao.data.
0009 %
0010 % CALL: ao_out = sin(ao_in);
0011 %       ao_out = sin(ao_in, pl);
0012 %       ao_out = sin(ao1, pl1, ao_vector, ao_matrix, pl2);
0013 %
0014 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0015 %                   ao_in  = ao_vector
0016 %                   ao_in  = ao_matrix
0017 %
0018 % PARAMETER LIST:  <key>          <value>          <description>
0019 %                   tsdata  fsdata  xydata
0020 %                   'xdata'  't'      'f'      'x'      compute the xdata
0021 %                   'ydata'  'x'      'xx'     'y'      compute the ydata
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 % >> pl = sin(ao, 'Params')
0027 %

```

```

0028 % VERSION: $Id: sin.m,v 1.17 2007/11/26 14:48:13 ingo Exp $
0029 %
0030 % HISTORY: 07-05-2007 Diepholz
0031 %           Creation
0032 %
0033 % SEE ALSO: ao/cos, ao/tan, ao/asin, acos, atan, Contents
0034 %
0035 % SEE: <a href="matlab: help ao/Contents;">HELP: All Analysis Object Functions:
</a>
0036 %           <a href="matlab: helpwin ao/Contents;">DOC: All Analysis Object
Functions:</a>
0037 %
0038 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0039
0040 VERSION = '$Id: sin.m,v 1.17 2007/11/26 14:48:13 ingo Exp $';
0041 ao_out   = [];
0042
0043 %% Check if this is a call for parameters
0044 if nargin == 2
0045     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0046         in = char(varargin{2});
0047         if strcmp(in, 'Params')
0048             ao_out = getDefaultPL();
0049             return
0050         elseif strcmp(in, 'Version')
0051             ao_out = VERSION;
0052             return
0053         end
0054     end
0055 end
0056
0057 %% Collect input ao's, plist's and ao variable names
0058 in_names = {};
0059 for ii = 1:nargin
0060     in_names{end+1} = inputname(ii);
0061 end
0062
0063 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0064
0065 if ~isempty(pl)
0066     pl = combine(pl);
0067 end
0068
0069 %% go through analysis objects
0070 for j=1:numel(ao_set)
0071     a = ao_set(j);
0072
0073     [h, sin_data] = single_operation(a.data, 'sin', pl);
0074
0075     %% Add the history from the ao object to the history
0076     h = set(h, 'inhists', [a.hist]);
0077

```



```
0078 %% Set the var_name to the history
0079 h = set(h, 'invars', cellstr(invars{j}));
0080
0081 %% create a new analysis objects
0082 new_ao = ao (sin_data, h);
0083 new_ao = setnh(new_ao, 'name', sprintf('sin(%s)',invars{j}) );
0084
0085 ao_out = [ao_out new_ao];
0086
0087 end
0088
0089 % Reshape the ouput to the same size of the input
0090 ao_out = reshape(ao_out, size(ao_set));
0091
0092
0093 %% Get default params
0094 function pl_default = getDefaultPL()
0095
0096     pl_default = plist([param('xdata', '')
0097                          param('ydata', '')]);
0098
0099
0100 % END
```

[Home](#) > [classes](#) > [@ao](#) > spectrogram.m

spectrogram

PURPOSE

SPECTROGRAM computes a spectrogram of the given ao/tsdata.

SYNOPSIS

function varargout = spectrogram(varargin)

DESCRIPTION

SPECTROGRAM computes a spectrogram of the given ao/tsdata.

%%%

DESCRIPTION: **SPECTROGRAM** computes a spectrogram of the given ao/tsdata using MATLAB's spectrogram function.

CALL: b = spectrogram(a, pl)

PARAMETERS:

 'Win' - a specwin object [default: Kaiser -200dB psll]
 'Nolap' - segment overlap [default: taken from window function]
 'Nfft' - number of samples in each short fourier transform
 [default: sample rate of data]

VERSION: \$Id: spectrogram.m,v 1.1 2007/12/26 19:20:20 hewitson Exp \$

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

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

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.
- **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.
- **spectrogram** SPECTROGRAM computes a spectrogram of the given ao/tsdata.

This function is called by:

- **spectrogram** SPECTROGRAM computes a spectrogram of the given ao/tsdata.

SUBFUNCTIONS

- **function plo = getDefaultPL()**

SOURCE CODE

```

0001 function varargout = spectrogram(varargin)
0002 % SPECTROGRAM computes a spectrogram of the given ao/tsdata.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SPECTROGRAM computes a spectrogram of the given ao/tsdata
0007 %               using MATLAB's spectrogram function.
0008 %
0009 % CALL:         b = spectrogram(a, pl)
0010 %
0011 % PARAMETERS:
0012 %
0013 %           'Win'   - a specwin object [default: Kaiser -200dB psll]
0014 %           'Nolap' - segment overlap [default: taken from window function]
0015 %           'Nfft'  - number of samples in each short fourier transform
0016 %                   [default: sample rate of data]
0017 %
0018 % VERSION:      $Id: spectrogram.m,v 1.1 2007/12/26 19:20:20 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = spectrogram(ao, 'Params')
0024 %
0025 % HISTORY: 12-03-07 M Hewitson
0026 %           Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029 %
0030 ALGONAME = mfilename;
0031 VERSION  = '$Id: spectrogram.m,v 1.1 2007/12/26 19:20:20 hewitson Exp $';
0032 bs       = [];
0033
0034 % Check if this is a call for parameters

```

```

0035 if nargin == 2
0036     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0037         in = char(varargin{2});
0038         if strcmp(in, 'Params')
0039             varargout{1} = getDefaultPL();
0040             return
0041         elseif strcmp(in, 'Version')
0042             varargout{1} = VERSION;
0043             return
0044         end
0045     end
0046 end
0047
0048 % Collect input ao's, plist's and ao variable names
0049 in_names = {};
0050 for ii = 1:nargin
0051     in_names{end+1} = inputname(ii);
0052 end
0053
0054 [as, ps, invars] = collect_inputs(varargin, in_names);
0055
0056 % Process parameters
0057 pl = combine(ps, getDefaultPL);
0058
0059 % Check input analysis object
0060 for j=1:numel(as)
0061     a      = as(j);
0062
0063     % Get settings for this AO
0064     nfft = find(pl, 'Nfft');
0065     if isempty(nfft) || nfft < 0
0066         nfft = a.data.fs;
0067     end
0068     win  = find(pl, 'Win');
0069     if length(win.win) < nfft
0070         switch win.name
0071             case {'Kaiser', 'Flattop'}
0072                 win = specwin(win.name, nfft, win.psl1);
0073             otherwise
0074                 win = specwin(win.name, nfft);
0075         end
0076         disp(sprintf('! Reset window to %s(%d)', strrep(win.name, '_', '\_'), length
(win.win)))
0077     end
0078     nolak = find(pl, 'Nolak');
0079     if isempty(nolak) || nolak < 0
0080         nolak = floor(win.rov*nfft/100);
0081     end
0082
0083 % Process data
0084 [S, F, T, P] = spectrogram(a.data.y, win.win, nolak, nfft, a.data.fs);
0085

```

```

0086 % make a new history object
0087 h = history(ALGONAME, VERSION, pl, [a.hist]);
0088 h = set(h, 'invars', cellstr(invars{j}));
0089
0090 % Make output A0
0091 do = xyzdata(T, F, P);
0092 do = set(do, 'xunits', 's');
0093 do = set(do, 'yunits', 'Hz');
0094 do = set(do, 'zunits', [a.data.yunits '^2/Hz']);
0095
0096 b = ao(do);
0097 b = setnh(b, 'name', sprintf('specgram(%s)', invars{j}));
0098
0099 % add to output
0100 bs = [bs b];
0101 end
0102
0103 % Reshape the ouput to the same size of the input
0104 bs = reshape(bs, size(as));
0105
0106 varargout{1} = bs;
0107
0108
0109 %-----
0110 % Get default params
0111 function plo = getDefaultPL()
0112
0113 w = specwin('Kaiser', 10, 200);
0114 plo = plist('Win', w, 'Nolap', -1, 'Nfft', -1);
0115
0116 % END

```

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.

%%%

DESCRIPTION: SPLIT split an analysis object into the specified segments.

CALL: b = split(a, pl)

INPUTS: a - input analysis object
 pl - input parameter list (see below for parameters)

OUTPUTS: b - array of analysis objects

PARAMETERS:	<key>	<value>	<description>
	'split_type'	'times'	split the ao into time segments
		'frequencies'	split the ao into frequencies segments
		'samples'	split the ao into sample segments
		'chunks'	split the ao into chunks
		'duration'	select a duration of a tsdata

Necessary for the individual split types:

by	<split type>	<key>	<description>
	'times'	'times'	an array of start/stop times to split by
	'frequencies'	'frequencies'	an array of start/stop frequencies to split by
	'samples'	'samples'	an array of start/stop samples to split by
	'chunks'	'N'	split into N contiguous pieces
	'interval'	'start_time', 'stop_time'	
		- start/stop time can be either a string or a time object (if a string, 'format_str' must be specified or specify the strings in UTC time format)	
	'interval'	'start_time', 'duration'	
		- start time and the duration can be either a string or a time object (if a string, 'format_str' must be specified or specify the strings in UTC time format)	

```
'interval'      'timespan'
                - the start/end time are specified in the time span
                  object.
```

The UTC time format is: 'yyyy-mm-dd HH:MM:SS'

If more than one splitting method is specified, the priority goes like the list above.

The time vector in the output A0 retains the original time values (i.e. it doesn't start from zero).

The splitting is done as $s \leq t < e$.

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')
```

```
VERSION:      $Id: split.m,v 1.29 2008/01/20 15:39:09 mauro 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.
- [setnh](#) SET set an analysis object property without recording the history.
- [times](#) TIMES overloads .* operator for analysis objects.

This function is called by:

SUBFUNCTIONS [^](#)

- [function plo = getDefaultPL\(\)](#)
- [function bool = isUTCformat\(time_string\)](#)
- [function sec = getSeconds\(time_str, format_str\)](#)

SOURCE CODE [^](#)

```

0001 function bo = split(varargin)
0002 % SPLIT split an analysis object into the specified segments.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SPLIT split an analysis object into the specified segments.
0007 %

```



```

0008 % CALL:      b = split(a, pl)
0009 %
0010 % INPUTS:      a - input analysis object
0011 %              pl - input parameter list (see below for parameters)
0012 %
0013 % OUTPUTS:      b - array of analysis objects
0014 %
0015 % PARAMETERS: <key>      <value>      <description>
0016 %              'split_type' 'times'      split the ao into time segments
0017 %              'frequencies' 'frequencies' split the ao into frequencies
0018 %              'samples'      split the ao into sample segments
0019 %              'chunks'      split the ao into chunks
0020 %              'duration'     select a duration of a tsdata
0021 %
0022 % Necessary for the individual split types:
0023 %              <split type> <key>      <description>
0024 %              'times'      'times'      an array of start/stop times to split
0025 %              'frequencies' 'frequencies' an array of start/stop frequencies to
0026 %              'samples'      'samples'      an array of start/stop samples to
0027 %              'chunks'      'N'          split into N contiguous pieces
0028 %              'interval'    'start_time', 'stop_time'
0029 %              - start/stop time can be either a string or a time
0030 %              object (if a string, 'format_str' must be
0031 %              specified or specify the strings in UTC time
0032 %              'interval'    'start_time', 'duration'
0033 %              - start time and the duration can be either a string
0034 %              or a time object (if a string, 'format_str' must
0035 %              be
0036 %              specified or specify the strings in UTC time
0037 %              'interval'    'timespan'
0038 %              - the start/end time are specified in the time span
0039 %              object.
0040 %
0041 %              The UTC time format is: 'yyyy-mm-dd HH:MM:SS'
0042 %
0043 %              If more than one splitting method is specified, the priority
0044 %              goes like the list above.
0045 %
0046 %              The time vector in the output A0 retains the original
0047 %              time values (i.e. it doesn't start from zero).
0048 %
0049 %              The splitting is done as s<=t<e.
0050 %
0051 %              Arrays of start/stop values should be like: [s1 e1 s2 e2 ....]
0052 %
0053 % EXAMPLES:      1.) Split method by frequency. Get the values from 10-100 Hz
                   pl = plist('split_type', 'frequencies', ...

```

```

0054 %             'frequencies', [10 100]);
0055 %             ao_new = split(a1, pl);
0056 %
0057 % 2.) Split method by time.
0058 %     Get the values from 0.0 to 1.0 Seconds AND from 1.0 to 2.5
seconds
0059 %             pl = plist('split_type', 'times', ...
0060 %                 'times',      [0.0 1.0 1.0 2.5]);
0061 %             ao_new = split(a1, pl);
0062 %
0063 % 3.) Split method by samples.
0064 %     Get the samples from 0 to 50 AND from 150 to 200.
0065 %             pl = plist('split_type', 'samples', ...
0066 %                 'samples',    [0 50 150 200]);
0067 %             ao_new = split(a1, pl);
0068 %
0069 % 4.1) Select an interval with strings
0070 %             pl = plist('split_type', 'interval', ...
0071 %                 'start_time', '14:00:01', ...
0072 %                 'end_time',   '14:00:02', ...
0073 %                 'format_str', 'HH:MM:SS');
0074 %             ao_new = split(a1, pl);
0075 %
0076 %             pl = plist('split_type', 'interval', ...
0077 %                 'start_time', '14:00:01', ...
0078 %                 'duration',   '00:00:02', ...
0079 %                 'format_str', 'HH:MM:SS');
0080 %             ao_new = split(a1, pl);
0081 %
0082 % 4.2) Select an interval with time objects
0083 %             pl = plist('split_type', 'interval', ...
0084 %                 'start_time', time('14:00:01', 'HH:MM:SS'), ...
0085 %                 'end_time',   time('2007-10-10 14:00:03'));
0086 %             ao_new = split(a1, pl);
0087 %
0088 %             pl = plist('split_type', 'interval', ...
0089 %                 'start_time', time('14:00:01', 'HH:MM:SS'), ...
0090 %                 'duration',   time('00:00:05', 'HH:MM:SS'));
0091 %             ao_new = split(a1, pl);
0092 %
0093 % 4.3) Select an interval with a time span object
0094 %             pl = plist('split_type', 'interval', ...
0095 %                 'timespan', timespan('14:00:00', '14:00:05', 'HH:
MM:SS'));
0096 %             ao_new = split(a1, pl);
0097 %
0098 % The following call returns a parameter list object that
0099 % contains the default parameter values:
0100 % >> pl = split(ao, 'Params')
0101 %
0102 % VERSION:      $Id: split.m,v 1.29 2008/01/20 15:39:09 mauro Exp $
0103 %
0104 % HISTORY: 02-03-07 M Hewitson

```

```

0105 %           Creation.
0106 %
0107 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0108
0109 ALGONAME = 'split';
0110 VERSION  = '$Id: split.m,v 1.29 2008/01/20 15:39:09 mauro Exp $';
0111 bo       = [];
0112
0113 % Check if this is a call for parameters
0114 if nargin == 2
0115     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0116         in = char(varargin{2});
0117         if strcmp(in, 'Params')
0118             bo = getDefaultPL();
0119             return
0120         elseif strcmp(in, 'Version')
0121             bo = VERSION;
0122             return
0123         end
0124     end
0125 end
0126
0127 %% Collect input ao's, plist's and ao variable names
0128 in_names = {};
0129 for ii = 1:nargin
0130     in_names{end+1} = inputname(ii);
0131
0132     if isnumeric(varargin{ii})
0133         samples = [samples varargin{ii}];
0134     end
0135 end
0136
0137 [as, pl, invars] = collect_inputs(varargin, in_names);
0138
0139 if ~isempty(pl)
0140     pl = combine(pl);
0141 end
0142
0143 %% go through analysis objects
0144 for j=1:numel(as)
0145
0146     % Unpack parameter list
0147     split_type = find(pl, 'split_type');
0148
0149     % check old version without the key 'split_type'
0150     if ~isempty(find(pl, 'samples'))
0151         split_type = 'samples';
0152     elseif ~isempty(find(pl, 'times')) || ...
0153         ~isempty(find(pl, 'frequencies'))
0154         split_type = 'times'; % times is the same to frequencies
0155     end
0156

```

```

0157 if isempty(split_type)
0158     error('### please specify the key ''split_type'' in the parameter list');
0159 end
0160
0161 % look at input data
0162 d = as(j).data;
0163
0164 [x,y] = get_xy_values(as(j).data);
0165
0166 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
0167 %                               splitting by time or frequency
%
0168 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
0169 if strcmpi(split_type, 'times') || strcmpi(split_type, 'frequencies')
0170
0171     times        = find(pl, 'times');
0172     frequencies = find(pl, 'frequencies');
0173
0174     if ~isempty(times)
0175         disp('* splitting by time...');
0176         split_x_axis.type = 'times';
0177         split_x_axis.value = times;
0178     else
0179         disp('* splitting by frequency...');
0180         split_x_axis.type = 'frequencies';
0181         split_x_axis.value = frequencies;
0182     end
0183
0184 % examine time list
0185 ntimes = length(split_x_axis.value);
0186 if mod(ntimes, 2) ~= 0
0187     error('### please specify a start and stop for each interval.')
0188 end
0189
0190 % go over each interval now
0191 for i=1:2:ntimes
0192
0193     is = split_x_axis.value(i);
0194     ie = split_x_axis.value(i+1);
0195
0196     idx = find(x>=is & x <ie);
0197
0198     % Make output analysis object
0199     nameStr = sprintf('split(%s)', d.name);
0200
0201     % create new output data
0202     d = set_xy_axis(d, x(idx), y(idx));
0203     d = set(d, 'name', nameStr);
0204
0205     % create new output history

```

```

0206     h = history(ALGONAME, VERSION, plist(param(split_x_axis.type, [is ie]),
as(j).hist);
0207     h = set(h, 'invars', invars(j));
0208
0209     % Set nsecs for tsdata
0210     if isa(d, 'tsdata')
0211         if ~isempty(d.x)
0212             d.nsecs = d.x(end) - d.x(1) + 1/d.fs;
0213         else
0214             d.nsecs = 0;
0215         end
0216     end
0217
0218     % make output analysis object
0219     b = ao(d, h);
0220
0221     % set name
0222     b = setnh(b, 'name', sprintf('split(%s)', invars{j}));
0223
0224     % Add to output array
0225     bo = [bo b];
0226
0227 end
0228
0229 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0230 %                                splitting by
samples                                %
0231 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0232 elseif strcmpi(split_type, 'samples')
0233
0234     disp('* splitting by samples...');
0235
0236     samples = find(pl, 'samples');
0237
0238     % examine time list
0239     npairs = length(samples);
0240     if mod(npairs, 2) ~= 0
0241         error('### please specify a start and stop for each interval.')
0242     end
0243
0244     % go over each interval now
0245     for i=1:2:npairs
0246
0247         is = samples(i);
0248         ie = samples(i+1);
0249
0250         % Make output analysis object
0251         nameStr = sprintf('split(%s)', d.name);
0252
0253         % create new output data

```

```

0254     x1 = x(is:ie);
0255     y1 = y(is:ie);
0256     d = set_xy_axis(d, x1, y1);
0257     d = set(d, 'name', nameStr);
0258
0259     % create new output history
0260     h = history(ALGONAME, VERSION, plist(param('samples', [is ie])), as(j).
hist);
0261     h = set(h, 'invars', invars);
0262
0263     % make output analysis object
0264     b = ao(d, h);
0265
0266     % set name
0267     b = setnh(b, 'name', sprintf('split(%s)', invars{j}));
0268
0269     % Add to output array
0270     bo = [bo b];
0271
0272 end
0273
0274 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
0275 %                               splitting into
chunks                               %
0276 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
0277 elseif strcmpi(split_type, 'chunks')
0278
0279     N = find(pl, 'N');
0280
0281     disp(sprintf('* splitting into %d chunks', N));
0282     error('### I have not been written yet. Please code me up.');
```

```

0283
0284 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
0285 %                               splitting into
interval                               %
0286 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
0287 elseif strcmpi(split_type, 'interval')
0288
0289     %% get values from the parameter list
0290     duration = find(pl, 'duration');
0291     start_time = find(pl, 'start_time');
0292     end_time = find(pl, 'end_time');
0293     format_str = find(pl, 'format_str');
0294     timespan = find(pl, 'timespan');
```

```

0295
0296 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Convert the time object %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
0297 if ~isempty(start_time) && isa(start_time, 'time')
```

```

0298     if ~isempty(format_str)
0299         start_time = set(start_time, 'timeformat', format_str);
0300     else
0301         format_str = char(start_time.timeformat);
0302     end
0303     %     format_str = char(start_time.timeformat);
0304     start_time = char(start_time);
0305 end
0306
0307 if ~isempty(end_time) && isa(end_time, 'time')
0308     if ~isempty(format_str)
0309         end_time = set(end_time, 'timeformat', format_str);
0310     else
0311         format_str = char(end_time.timeformat);
0312     end
0313     %     format_str = char(end_time.timeformat);
0314     end_time = char(end_time);
0315 end
0316
0317 if ~isempty(duration) && isa(duration, 'time')
0318     if ~isempty(format_str)
0319         duration = set(duration, 'timeformat', format_str);
0320     else
0321         format_str = char(duration.timeformat);
0322     end
0323     %     format_str = char(end_time.timeformat);
0324     duration = char(duration);
0325 end
0326
0327 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Convert the time span object %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0328 %
0329 if ~isempty(timespan)
0330     start_time = char(timespan.start);
0331     end_time = char(timespan.end);
0332     format_str = char(timespan.timeformat);
0333     duration = '';
0334 end
0335
0336 UTC = false;
0337
0338 %% check the input values
0339 if ~isa(as(j).data, 'tsdata')
0340     error('### the split type: ''duration'' only works with tsdata.')
0341 end
0342
0343 if (isempty(start_time) && ...
0344     (isempty(duration) || isempty(end_time)))
0345     error('### please specify the key ''start_time'' AND ''end_stime'' OR
0346         ''duration'' in the parameter list')
0347 end
0348
0349

```

```

0347     if (ischar(start_time) || ...
0348         ischar(end_time) || ...
0349         ischar(duration) ) && ...
0350         isempty(format_str)
0351         %% are the time strings in UTC (yyy-mm-dd HH:MM:SS)
0352         %% then set the format string to the UTC format.
0353         if isUTCformat(start_time) || ...
0354             isUTCformat(end_time) || ...
0355             isUTCformat(duration)
0356             format_str = 'yyyy-mm-dd HH:MM:SS';
0357         else
0358             error ([char (10) '### please specify the key ''format_str'' for the
keys ' ...
0359                 ''start_time'', ''end_time'' OR/AND ''duration'''])
0360         end
0361     end
0362
0363     disp('* splitting by interval ...');
0364
0365     %% convert the start time
0366     if isUTCformat(start_time)
0367         UTC = true;
0368         start_time = time(start_time);
0369         start_time = start_time.utc_epoch_milli/1000;
0370     elseif ischar (start_time)
0371         start_time = getSeconds(start_time, format_str);
0372     end
0373
0374     %% convert the end time
0375     if isUTCformat(end_time)
0376         UTC = true;
0377         end_time = time(end_time);
0378         end_time = end_time.utc_epoch_milli/1000;
0379     elseif ischar (end_time)
0380         end_time = getSeconds(end_time, format_str);
0381     end
0382
0383     %% convert the duration
0384     if ~isempty(end_time) && ~isempty(duration)
0385         warning ([ '### in the parameter list are both parameter ''...
0386                 ''end_time'' and ''duration''. Using the parameter ''end_time'''])
0387     elseif ischar(duration) && ~isempty(duration)
0388         duration = getSeconds(duration, format_str);
0389     end
0390
0391     %% get/convert the t0
0392     t0 = as(j).data.t0;
0393     t_unix_time = t0.utc_epoch_milli/1000;
0394     t1 = x(1);
0395
0396     if UTC == true
0397         ts = start_time-t_unix_time+t1;

```



```

0398     if ~isempty(end_time)
0399         te = end_time-t_unix_time+t1;
0400     else
0401         te = ts+duration;
0402     end
0403 else
0404
0405     t0_offset = format(t0, format_str);
0406     t0_offset = getSeconds(t0_offset, format_str);
0407
0408     ts = start_time+t1-t0_offset;
0409     if ~isempty(end_time)
0410         te = end_time+t1-t0_offset;
0411     else
0412         te = start_time+duration+t1-t0_offset;
0413     end
0414 end
0415
0416 idx = find(x>=ts & x <te);
0417
0418 % Make output analysis object
0419 nameStr = sprintf('split(%s)', d.name);
0420
0421 % create new output data
0422 d = set_xy_axis(d, x(idx), y(idx));
0423 d = set(d, 'name', nameStr);
0424
0425 % Set nsecs for tsdata
0426 if isa(d, 'tsdata')
0427     if ~isempty(d.x)
0428         d.nsecs = d.x(end) - d.x(1) + 1/d.fs;
0429     else
0430         d.nsecs = 0;
0431     end
0432 end
0433
0434 % create new output history
0435 h = history(ALGONAME, VERSION, pl, as(j).hist);
0436 h = set(h, 'invars', invars);
0437
0438 % make output analysis object
0439 b = ao(d, h);
0440
0441 % set name
0442 b = setnh(b, 'name', sprintf('split(%s)', invars{j}));
0443
0444 % Add to output array
0445 bo = [bo b];
0446
0447
0448 else
0449     error('### do not know how to split.')

```

```

0450     end
0451
0452 end % for j = 1:numel(as)
0453
0454 % Reshape the ouput to the same size of the input
0455 if numel(bo) == numel(as)
0456     bo = reshape(bo, size(as));
0457 end
0458
0459 end % split
0460
0461 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0462 %
0463 % FUNCTION:      getDefaultPL
0464 %
0465 % DESCRIPTION: Returns the default parameter list
0466 %
0467 % HISTORY:       02-03-07 M Hewitson
0468 %                Creation.
0469 %
0470 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0471
0472 function plo = getDefaultPL()
0473
0474     disp('* creating default plist...');
0475     plo = plist();
0476     plo = append(plo, param('split_type', ''));
0477     plo = append(plo, param('times', []));
0478     plo = append(plo, param('frequencies', []));
0479     plo = append(plo, param('samples', []));
0480     plo = append(plo, param('N', []));
0481     plo = append(plo, param('duration', ''));
0482     plo = append(plo, param('start_time', ''));
0483     plo = append(plo, param('end_time', ''));
0484     plo = append(plo, param('format_str', ''));
0485     disp('* done. ');
0486
0487 end % getDefaultPL
0488
0489 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0490 %
0491 % FUNCTION:      isUTCformat
0492 %
0493 % DESCRIPTION: Check the input to the UTC time format (yyyy-mm-dd- HH:MM:SS)
0494 %
0495 % HISTORY:       11-07-07 Diepholz
0496 %                Creation.
0497 %
0498 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0499
0500 function bool = isUTCformat(time_string)
0501
0502     bool = false;

```

```

0503
0504     if ischar(time_string)
0505         if regexp(time_string, '\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}')
0506             bool = true;
0507         end
0508     end
0509
0510 end % isUTCformat
0511
0512
0513 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0514 %
0515 % FUNCTION:      getSeconds
0516 %
0517 % DESCRIPTION: Returns the seconds for any time format with hours minutes
0518 %               ans seconds.
0519 %
0520 % HISTORY:      11-07-07 Diepholz
0521 %               Creation.
0522 %
0523 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0524
0525 function sec = getSeconds(time_str, format_str)
0526
0527 % The customer use year, month or day to specify the time_str
0528 if regexp(format_str, '[ymd]')
0529     if ~strcmp('yyyy-mm-dd HH:MM:SS', format_str)
0530         error (['### if you want to specify the time without using the UTC time
0531 ...
0532         'use only hours(HH) minutes(MM) and seconds(SS)'])
0533     end
0534 end
0535 try
0536     sec = ( datenum(time_str, format_str) - fix(datenum(time_str,
0537 format_str)) ) * 86400;
0538     sec = round(sec*1000)/1000;
0539 catch
0540     last_error = lasterror;
0541     if strfind(last_error.identifier, 'datenum')
0542         error ([last_error.message char(10) char(10) ...
0543             'time_str = '' time_str '' format string = '' format_str '''])
0544     else
0545         error (last_error.message)
0546     end
0547 end
0548 end % getSeconds

```

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')

HISTORY: 24-05-2007 Diepholz
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

```

0001 function ao_out = sqrt (varargin)
0002 % Sqrt overloads the sqrt operator for Analysis objects. Square root.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: Sqrt overloads the sqrt operator for Analysis objects.
0007 %               Square root.
0008 %               Sqrt(ao) is the square root of the elements of ao.data.
0009 %
0010 % CALL: ao_out = sqrt(ao_in);
0011 %       ao_out = sqrt(ao_in, pl);
0012 %       ao_out = sqrt(ao1, pl1, ao_vector, ao_matrix, pl2);
0013 %
0014 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0015 %                   ao_in  = ao_vector
0016 %                   ao_in  = ao_matrix
0017 %
0018 % PARAMETER LIST:  <key>          <value>          <description>
0019 %                   tsdata  fsdata  xydata
0020 %                   'xdata'  't'      'f'      'x'      compute the xdata
0021 %                   'ydata'  'x'      'xx'     'y'      compute the ydata
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 % >> pl = sqrt(ao, 'Params')
0027 %
0028 % HISTORY: 24-05-2007 Diepholz
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032 %
0033 VERSION  = '$Id: sqrt.m,v 1.14 2007/11/26 14:48:13 ingo Exp $';
0034 ao_out   = [];
0035

```

```

0036 %% Check if this is a call for parameters
0037 if nargin == 2
0038     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0039         in = char(varargin{2});
0040         if strcmp(in, 'Params')
0041             ao_out = getDefaultPL();
0042             return
0043         elseif strcmp(in, 'Version')
0044             ao_out = VERSION;
0045             return
0046         end
0047     end
0048 end
0049
0050 %% Collect input ao's, plist's and ao variable names
0051 in_names = {};
0052 for ii = 1:nargin
0053     in_names{end+1} = inputname(ii);
0054 end
0055
0056 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0057
0058 if ~isempty(pl)
0059     pl = combine(pl);
0060 end
0061
0062 %% go through analysis objects
0063 for j=1:numel(ao_set)
0064     a = ao_set(j);
0065
0066     [h, sqrt_data] = single_operation(a.data, 'sqrt', pl);
0067
0068     %% Add the history from the ao object to the history
0069     h = set(h, 'inhists', [a.hist]);
0070
0071     %% Set the var_name to the history
0072     h = set(h, 'invars', cellstr(invars{j}));
0073
0074     %% create a new analysis objects
0075     new_ao = ao (sqrt_data, h);
0076     new_ao = setnh(new_ao, 'name', sprintf('sqrt(%s)', invars{j}));
0077
0078     ao_out = [ao_out new_ao];
0079
0080 end
0081
0082 % Reshape the output to the same size of the input
0083 ao_out = reshape(ao_out, size(ao_set));
0084
0085 %% Get default params
0086 function pl_default = getDefaultPL()

```

```
0087
0088     pl_default = plist([param('xdata', '')
0089                          param('ydata', '')] );
0090
0091 % END
```

Generated on Tue 22-Jan-2008 10:39:13 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.

%%%

DESCRIPTION: STD overloads the std operator for Analysis objects.
Compute the standard deviation.
STD(ao) is the standard deviation value of the elements in ao.data.

CALL: ao_out = std(ao_in);
ao_out = std(ao_in, dim);
ao_out = std(ao_in, FLAG, DIM); --> use the parameterlist
ao_out = std(ao_in, pl);
ao_out = std(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'	0 or 1	represent the FLAG		
	'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')

HISTORY: 23-05-2007 Diepholz
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 pl_default = getDefaultPL\(\)](#)

SOURCE CODE

```

0001 function ao_out = std (varargin)
0002 % STD overloads the std operator for Analysis objects. Compute the standard
0003 % deviation.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: STD overloads the std operator for Analysis objects.
0007 %               Compute the standard deviation.
0008 %               STD(ao) is the standard deviation value of the elements in ao.
0009 % data.
0010 % CALL: ao_out = std(ao_in);
0011 %       ao_out = std(ao_in, dim);
0012 %       ao_out = std(ao_in, FLAG, DIM); --> use the parameterlist
0013 %       ao_out = std(ao_in, pl);
0014 %       ao_out = std(ao1, pl1, ao_vector, ao_matrix, pl2);
0015 %
0016 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0017 %                   ao_in  = ao_vector
0018 %                   ao_in  = ao_matrix
0019 %
0020 % PARAMETER LIST:  <key>          <value>          <description>
0021 %                  'w'           0 of 1           represent the FLAG
0022 %                  'dim'         1 or 2 or 3 ...    takes the std along the
0023 %                                     dimension dim
0024 %                                     tsdata  fsdata  xydata
0025 %                  'xdata'      't'         'f'         'x'         compute the std of the x-
0026 % axis                          'ydata'      'x'         'xx'        'y'         compute the std of the y-
0027 % axis

```

```

0028 % The following call returns a parameter list object that contains the
0029 % default parameter values:
0030 %
0031 % >> pl = std(ao, 'Params')
0032 %
0033 % HISTORY: 23-05-2007 Diepholz
0034 %           Creation
0035 %
0036 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0037
0038 VERSION = '$Id: std.m,v 1.9 2007/11/26 14:48:13 ingo Exp $';
0039 ao_out = [];
0040
0041 %% Check if this is a call for parameters
0042 if nargin == 2
0043     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0044         in = char(varargin{2});
0045         if strcmp(in, 'Params')
0046             ao_out = getDefaultPL();
0047             return
0048         elseif strcmp(in, 'Version')
0049             ao_out = VERSION;
0050             return
0051         end
0052     end
0053 end
0054
0055 %% Collect input ao's, plist's and ao variable names
0056 in_names = {};
0057 ps = plist();
0058 for ii = 1:nargin
0059     in_names{end+1} = inputname(ii);
0060
0061     if isnumeric(varargin{ii})
0062         ps = [ps plist(param('dim', varargin{ii}))];
0063     end
0064 end
0065
0066 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0067
0068 pl = combine(pl, ps);
0069
0070 %% go through analysis objects
0071 for j=1:numel(ao_set)
0072     a = ao_set(j);
0073
0074     [h, std_data] = single_operation(a.data, 'std', pl);
0075
0076     %% Add the history from the ao object to the history
0077     h = set(h, 'inhists', [a.hist]);
0078
0079     %% Set the var_name to the history

```

```

0080     h = set(h, 'invars', cellstr(invars{j}));
0081
0082     % convert to cdata type
0083     std_x     = [];
0084     std_y     = [];
0085
0086     do_xdata = find(pl, 'xdata');
0087     do_ydata = find(pl, 'ydata');
0088
0089     %% Is no axis entry in the parameter list
0090     %% then set the default axis = xdata
0091     if isempty(do_xdata) && isempty(do_ydata)
0092         do_ydata = 'yes';
0093     end
0094
0095     %% Get the operation result. It is stored in the individual axis
0096     pl_get_axis = plist([param('xdata', do_xdata) param('ydata', do_ydata)]);
0097     [std_x, std_y] = get_xy_values(std_data, pl_get_axis);
0098
0099     std_cdata = cdata([std_x std_y]);
0100     std_cdata = set (std_cdata, 'name', std_data.name);
0101     std_cdata = set (std_cdata, 'xunits', std_data.xunits);
0102     std_cdata = set (std_cdata, 'yunits', std_data.yunits);
0103
0104     %% create a new analysis objects
0105     new_ao = ao (std_cdata, h);
0106     new_ao = setnh(new_ao, 'name', sprintf('std(%s)', invars{j}) );
0107
0108     ao_out = [ao_out new_ao];
0109
0110 end
0111
0112 % Reshape the ouput to the same size of the input
0113 ao_out = reshape(ao_out, size(ao_set));
0114
0115 %% Get default params
0116 function pl_default = getDefaultPL()
0117
0118     pl_default = plist([param('dim',    '')
0119                           param('w',    [])
0120                           param('xdata', '')
0121                           param('ydata', '')[1]]);
0122
0123 % END

```

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')

HISTORY: 25-05-2007 Diepholz
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

```

0001 function ao_out = sum (varargin)
0002 % SUM overloads the sum operator for Analysis objects. Compute the sum value.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUM overloads the sum operator for Analysis objects.
0007 %                Compute the sum value.
0008 %                SUM(ao) is the sum value of the elements in ao.data.
0009 %
0010 % CALL: ao_out = sum(ao_in);
0011 %       ao_out = sum(ao_in, dim);
0012 %       ao_out = sum(ao_in, pl);
0013 %       ao_out = sum(ao1, pl1, ao_vector, ao_matrix, pl2);
0014 %
0015 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0016 %                   ao_in  = ao_vector
0017 %                   ao_in  = ao_matrix
0018 %
0019 % PARAMETER LIST:  <key>          <value>          <description>
0020 %                  'dim'          1 or 2 or 3 ...    takes the sum along the
0021 %                                           dimension dim
0022 %                  tsdata  fsdata  xydata
0023 %                  'xdata' 't'     'f'     'x'     compute the sum of the x-
axis
0024 %                  'ydata' 'x'     'xx'    'y'     compute the sum of the y-
axis
0025 %
0026 %                  The following call returns a parameter list object that
0027 %                  contains the default parameter values:
0028 %
0029 %                  >> pl = sum(ao, 'Params')

```

```

0030 %
0031 % HISTORY: 25-05-2007 Diepholz
0032 %           Creation
0033 %
0034 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0035
0036 VERSION = '$Id: sum.m,v 1.7 2007/11/26 14:48:13 ingo Exp $';
0037 ao_out   = [];
0038
0039 % Check if this is a call for parameters
0040 if nargin == 2
0041     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0042         in = char(varargin{2});
0043         if strcmp(in, 'Params')
0044             ao_out = getDefaultPL();
0045             return
0046         elseif strcmp(in, 'Version')
0047             ao_out = VERSION;
0048             return
0049         end
0050     end
0051 end
0052
0053 %% Collect input ao's, plist's and ao variable names
0054 in_names = {};
0055 ps       = plist();
0056 for ii = 1:nargin
0057     in_names{end+1} = inputname(ii);
0058
0059     if isnumeric(varargin{ii})
0060         ps = [ps plist(param('dim', varargin{ii}))];
0061     end
0062 end
0063
0064 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0065
0066 pl = combine(pl, ps);
0067
0068 %% go through analysis objects
0069 for j=1:numel(ao_set)
0070     a = ao_set(j);
0071
0072     [h, sum_data] = single_operation(a.data, 'sum', pl);
0073
0074     %% Add the history from the ao object to the history
0075     h = set(h, 'inhists', [a.hist]);
0076
0077     %% Set the var_name to the history
0078     h = set(h, 'invars', cellstr(invars{j}));
0079
0080     do_xdata = find(pl, 'xdata');

```

```

0081 do_ydata = find(pl, 'ydata');
0082
0083 %% Is no axis entry in the parameter list
0084 %% then set the default axis = xdata
0085 if isempty(do_xdata) && isempty(do_ydata)
0086     do_ydata = 'yes';
0087 end
0088
0089 %% Get the operation result. It is stored in the individual axis
0090 pl_get_axis = plist([param('xdata', do_xdata) param('ydata', do_ydata)]);
0091 [sum_x, sum_y] = get_xy_values(sum_data, pl_get_axis);
0092
0093 sum_cdata = cdata([sum_x sum_y]);
0094 sum_cdata = set (sum_cdata, 'name', sum_data.name);
0095 sum_cdata = set (sum_cdata, 'xunits', sum_data.xunits);
0096 sum_cdata = set (sum_cdata, 'yunits', sum_data.yunits);
0097
0098 %% create a new analysis objects
0099 new_ao = a;
0100 new_ao = ao (sum_cdata, h);
0101 new_ao = setnh(new_ao, 'name', sprintf('sum(%s)', invars{j}) );
0102
0103 ao_out = [ao_out new_ao];
0104
0105 end
0106
0107 % Reshape the ouput to the same size of the input
0108 ao_out = reshape(ao_out, size(ao_set));
0109
0110 %% Get default params
0111 function plo = getDefaultPL()
0112
0113 plo = plist();
0114 plo = append(plo, param('xdata', ''));
0115 plo = append(plo, param('ydata', ''));
0116 plo = append(plo, param('dim', []));
0117
0118
0119 % END

```

[Home](#) > [classes](#) > [@ao](#) > svd.m

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 submitted 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')
```

VERSION: \$Id: svd.m,v 1.13 2007/11/26 14:48:13 ingo Exp \$

HISTORY: 08-05-07A 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

```

0001 function varargout = svd(varargin)
0002 % SVD overloads the determinant function for Analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SVD overloads the determinant function for Analysis objects.
0007 %
0008 % CALL:          u          = svd (a,pl)
0009 %               [u, s]      = svd (a,pl)
0010 %               [u, s, v]   = svd (a,pl)
0011 %               [u, s, v]   = svd (a)
0012 %
0013 % INPUTS:        a      - input analysis object
0014 %               pl      - a parameter list
0015 %
0016 % OUTPUTS:       like matlab fct
0017 %
0018 % PARAMETERS:    'option' - a string or value that can be submitted i.e. 'econ'
0019 %                  to produce economy size decomposition
0020 %                  - options are the same as for the matlab function
0021 %
0022 %               The following call returns a parameter list object that
0023 %               contains the default parameter values:
0024 %
0025 %               >> pl = svd(ao, 'Params')
0026 %
0027 % VERSION:       $Id: svd.m,v 1.13 2007/11/26 14:48:13 ingo Exp $
0028 %
0029 % HISTORY:       08-05-07A Monsky
0030 %                  Creation.
0031 %
0032 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0033

```

```

0034 VERSION = '$Id: svd.m,v 1.13 2007/11/26 14:48:13 ingo Exp $';
0035
0036 U = [];
0037 S = [];
0038 V = [];
0039
0040 %% Check if this is a call for parameters
0041
0042 if nargin == 2
0043     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0044         in = char(varargin{2});
0045         if strcmp(in, 'Params')
0046             varargout{1} = getDefaultPL();
0047             return
0048         elseif strcmp(in, 'Version')
0049             varargout{1} = VERSION;
0050             return
0051         end
0052     end
0053 end
0054
0055 %% Collect input ao's, plist's and ao variable names
0056 in_names = {};
0057 for ii = 1:nargin
0058     in_names{end+1} = inputname(ii);
0059 end
0060
0061 [as, pl, invars] = collect_inputs(varargin, in_names);
0062
0063 if ~isempty(pl)
0064     pl = combine(pl);
0065 end
0066
0067 % check plist
0068 if ~isempty(pl)
0069     pl = combine(pl);
0070 end
0071
0072 % Check input analysis object
0073 for j=1:numel(as)
0074     a = as(j);
0075
0076     d = get(a, 'data');
0077     dinfo = whos('d');
0078
0079     % Which data type do we have
0080     dtype = dinfo.class;
0081
0082     udata = [];
0083     sdata = [];
0084     vdata = [];
0085

```

```

0086  switch dtype
0087      case 'cdata'
0088          disp('* DETERMINANT of cdata object');
0089          if nargin <= 1
0090              [h, udata] = single_operation(a.data, 'svd',pl);
0091          elseif nargin == 2
0092              [h, udata, sdata] = single_operation(a.data, 'svd',pl);
0093          elseif nargin == 3
0094              [h, udata, sdata, vdata] = single_operation(a.data, 'svd',pl);
0095          end
0096          %create analysis object(s)
0097          h = set(h, 'inhists', [a.hist]);
0098
0099          %% Set the var_name to the history
0100          h = set(h, 'invars', cellstr(invars{j}));
0101
0102          u = ao(udata, h);
0103          u = setnh(u, 'name', sprintf('u_svd(%s)', char(invars{j})));
0104          U = [U u];
0105          if ~isempty(sdata)
0106              s = ao(sdata, h);
0107              s = setnh(s, 'name', sprintf('s_svd(%s)', char(invars{j})));
0108              S = [S s];
0109          end
0110          if ~isempty(vdata)
0111              v = ao(vdata, h);
0112              v = setnh(v, 'name', sprintf('v_svd(%s)', char(invars{j})));
0113              V = [V v];
0114          end
0115
0116          case {'tsdata','fsdata','xydata'}
0117              error('### this function works for cdata type A0 only')
0118          otherwise
0119              error('### unknown data type.')
0120      end
0121
0122 end
0123
0124 % Reshape the ouput to the same size of the input
0125 U = reshape(U, size(as));
0126 varargout{1} = U;
0127
0128 if nargin > 1
0129     % Reshape the ouput to the same size of the input
0130     S = reshape(S, size(as));
0131     varargout{2} = S;
0132     if nargin > 2
0133         % Reshape the ouput to the same size of the input
0134         V = reshape(V, size(as));
0135         varargout{3} = V;
0136         if nargin > 3

```

```
0137     error('### wrong number of outputs')
0138     end
0139 end
0140 end
0141
0142 %% Get default params
0143 function plo = getDefaultPL()
0144
0145 plo = plist();
0146 plo = append(plo, 'option', []);
0147
0148 % END
```

Generated on Tue 22-Jan-2008 10:39:13 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.

%%%

DESCRIPTION: TAN overloads the tan operator for Analysis objects.
Tangent of argument in radians.
TAN(ao) is the tangent of the elements of ao.data.

CALL: ao_out = tan(ao_in);
ao_out = tan(ao_in, pl);
ao_out = tan(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 = tan(ao, 'Params')

HISTORY: 07-05-2007 Diepholz
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

```

0001 function ao_out = tan (varargin)
0002 % TAN overloads the tan operator for Analysis objects. Tangent of argument in
0003 % radians.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: TAN overloads the tan operator for Analysis objects.
0007 %             Tangent of argument in radians.
0008 %             TAN(ao) is the tangent of the elements of ao.data.
0009 %
0010 % CALL: ao_out = tan(ao_in);
0011 %       ao_out = tan(ao_in, pl);
0012 %       ao_out = tan(ao1, pl1, ao_vector, ao_matrix, pl2);
0013 %
0014 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0015 %                   ao_in  = ao_vector
0016 %                   ao_in  = ao_matrix
0017 %
0018 % PARAMETER LIST:  <key>          <value>          <description>
0019 %                   tsdata  fsdata  xydata
0020 %                   'xdata'  't'      'f'      'x'      compute the xdata
0021 %                   'ydata'  'x'      'xx'     'y'      compute the ydata
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 % >> pl = tan(ao, 'Params')
0027 %
0028 % HISTORY: 07-05-2007 Diepholz
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032 %
0033 VERSION  = '$Id: tan.m,v 1.10 2007/11/26 14:48:13 ingo Exp $';
0034 ao_out   = [];
0035
0036 %% Check if this is a call for parameters

```

```

0037 if nargin == 2
0038     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0039         in = char(varargin{2});
0040         if strcmp(in, 'Params')
0041             ao_out = getDefaultPL();
0042             return
0043         elseif strcmp(in, 'Version')
0044             ao_out = VERSION;
0045             return
0046         end
0047     end
0048 end
0049
0050 %% Collect input ao's, plist's and ao variable names
0051 in_names = {};
0052 for ii = 1:nargin
0053     in_names{end+1} = inputname(ii);
0054 end
0055
0056 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0057
0058 if ~isempty (pl)
0059     pl = combine(pl);
0060 end
0061
0062 %% go through analysis objects
0063 for j=1:numel(ao_set)
0064     a = ao_set(j);
0065
0066     [h, tan_data] = single_operation(a.data, 'tan', pl);
0067
0068     %% Add the history from the ao object to the history
0069     h = set(h, 'inhists', [a.hist]);
0070
0071     %% Set the var_name to the history
0072     h = set(h, 'invars', cellstr(invars{j}));
0073
0074     %% create a new analysis objects
0075     new_ao = a;
0076     new_ao = ao (tan_data, h);
0077     new_ao = setnh(new_ao, 'name', sprintf('tan(%s)', invars{j}) );
0078
0079     ao_out = [ao_out new_ao];
0080
0081 end
0082
0083 % Reshape the output to the same size of the input
0084 ao_out = reshape(ao_out, size(ao_set));
0085
0086 %% Get default params
0087 function pl_default = getDefaultPL()
0088

```

```
0089    pl_default = plist([param('xdata', '')
0090                          param('ydata', '')]);
0091
0092 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > times.m

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 1.11 2007/11/23 12:11:08 ingo Exp \$

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

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

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:

- **aosplit** SPLIT an analysis object into the specified segments.
- **split** SPLIT split an analysis object into the specified segments.

SUBFUNCTIONS

- **function plo = getDefaultPL()**

SOURCE CODE

```

0001 function b = times(varargin)
0002 % TIMES overloads .* operator for analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: TIMES overloads .* operator for analysis objects.
0007 %               This just does the same as times (.* ) for analysis objects.
0008 %
0009 % CALL:         a = a1.*5;
0010 %
0011 % VERSION:      $Id: times.m,v 1.11 2007/11/23 12:11:08 ingo Exp $
0012 %
0013 %               The following call returns a parameter list object that
0014 %               contains the default parameter values:
0015 %
0016 %               >> pl = times(ao, 'Params')
0017 %
0018 % HISTORY: 05-02-07 M Hewitson
0019 %               Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 ALGONAME = mfilename;
0024 VERSION  = '$Id: times.m,v 1.11 2007/11/23 12:11:08 ingo Exp $';
0025
0026 % Check if this is a call for parameters
0027 if nargin == 2
0028     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0029         in = char(varargin{2});
0030         if strcmp(in, 'Params')
0031             b = getDefaultPL();
0032             return
0033         elseif strcmp(in, 'Version')
0034             b = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 % capture input variable names
0041 invars = {};

```

```

0042 for j=1:nargin
0043     iname = inputname(j);
0044     if isempty(iname) && isnumeric(varargin{j})
0045         iname = num2str(varargin{j});
0046     elseif isempty(iname) && ischar(varargin{j})
0047         iname = varargin{j};
0048     end
0049     invars = [invars cellstr(iname)];
0050 end
0051
0052 op = '.*';
0053
0054 [a1,a2,do] = aooperate(varargin, op);
0055
0056 %----- create output AO
0057
0058 % make a new history object
0059 h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
0060 h = set(h, 'invars', invars);
0061
0062 % get names for output
0063 if isempty(char(invars{1}))
0064     n1 = a1.name;
0065 else
0066     n1 = char(invars{1});
0067 end
0068 if isempty(char(invars{2}))
0069     n2 = a2.name;
0070 else
0071     n2 = char(invars{2});
0072 end
0073
0074 % make output analysis object
0075 % op = op(2:end);
0076 b = ao(do, h);
0077 b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2));
0078
0079
0080 % Get default params
0081 function plo = getDefaultPL()
0082
0083 plo = plist();
0084
0085
0086 % END

```

[Home](#) > [classes](#) > [@ao](#) > transpose.m

transpose

PURPOSE

TRANPOSE overloads the .' operator for Analysis Objects.

SYNOPSIS

function varargout = transpose(varargin)

DESCRIPTION

TRANPOSE overloads the .' operator for Analysis Objects.

%%%

DESCRIPTION: TRANPOSE overloads the .' operator for Analysis Objects.

CALL: a = a1.' % only with data = cdata

VERSION: \$Id: transpose.m,v 1.12 2007/11/26 14:48:13 ingo Exp \$

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

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

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

```

0001 function varargout = transpose(varargin)
0002 % TRANSPOSE overloads the '.' operator for Analysis Objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: TRANSPOSE overloads the '.' operator for Analysis Objects.
0007 %
0008 % CALL:          a = a1.'      % only with data = cdata
0009 %
0010 % VERSION:       $Id: transpose.m,v 1.12 2007/11/26 14:48:13 ingo Exp $
0011 %
0012 %               The following call returns a parameter list object that
0013 %               contains the default parameter values:
0014 %
0015 %               >> pl = transpose(ao, 'Params')
0016 %
0017 % HISTORY: 08-05-07 A Monsky
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: transpose.m,v 1.12 2007/11/26 14:48:13 ingo Exp $';
0023 bs      = [];
0024
0025 %% Check if this is a call for parameters
0026
0027 if nargin == 2
0028     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0029         in = char(varargin{2});
0030         if strcmp(in, 'Params')
0031             varargout{1} = getDefaultPL();
0032             return
0033         elseif strcmp(in, 'Version')
0034             varargout{1} = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 %% Collect input ao's, plist's and ao variable names
0041 in_names = {};
0042 for ii = 1:nargin
0043     in_names{end+1} = inputname(ii);
0044 end

```

```

0045
0046 [as, ps, invars] = collect_inputs(varargin, in_names);
0047
0048 % check plist
0049 if isempty(ps)
0050     pl = getDefaultPL();
0051 else
0052     pl = combine(ps, getDefaultPL);
0053 end
0054
0055 %create empty plist
0056 plo = plist();
0057
0058 %% go through analysis objects
0059 for j=1:numel(as)
0060     a = as(j);
0061     d = get(a, 'data');
0062     dinfo = whos('d');
0063
0064     % Which data type do we have
0065     dtype = dinfo.class;
0066
0067     switch dtype
0068     case 'cdata'
0069         disp('* TRANSPOSE of cdata object');
0070         % make a new cdata object
0071         [h, c] = single_operation(d, 'transpose', pl);
0072
0073         % make output analysis object
0074         h = set(h, 'inhists', [a.hist]);
0075
0076         %% Set the var_name to the history
0077         h = set(h, 'invars', cellstr(invars{j}));
0078
0079         b = ao(c, h);
0080         b = setnh(b, 'name', sprintf('transpose(%s)', char(invars{j})));
0081
0082     case {'tsdata', 'fsdata', 'xydata'}
0083         error('### this function works for cdata type A0 only')
0084     otherwise
0085         error('### unknown data type.')
0086
0087     end
0088
0089     % add to output
0090     bs = [bs b];
0091 end
0092
0093 % Reshape the ouput to the same size of the input
0094 bs = reshape(bs, size(as));
0095 varargout{1} = bs;

```

```
0096
0097 %% Get default params
0098 function plo = getDefaultPL()
0099
0100 plo = plist();
0101
0102 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > upsample.m

upsample

PURPOSE

UPSAMPLE overloads upsample function for AOs.

SYNOPSIS

function bs = upsample(varargin)

DESCRIPTION

UPSAMPLE overloads upsample function for AOs.

%%%

DESCRIPTION: UPSAMPLE overloads upsample function for AOs.

A signal at sample rate fs is upsampled by inserting N-1 zeros between the input samples.

CALL: b = upsample(a, pl)

Parameters: 'N' - specify the desired upsample rate.
'phase' - specify an initial phase in range [0, N-1].

VERSION: \$Id: upsample.m,v 1.2 2007/11/26 14:48:13 ingo Exp \$

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

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

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.

- **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

```

0001 function bs = upsample(varargin)
0002 % UPSAMPLE overloads upsample function for AOs.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: UPSAMPLE overloads upsample function for AOs.
0007 %
0008 %           A signal at sample rate fs is upsampled by inserting N-1 zeros
between the           input samples.
0009 %
0010 %
0011 % CALL:      b = upsample(a, pl)
0012 %
0013 % Parameters: 'N'      - specify the desired upsample rate.
0014 %           'phase' - specify an initial phase in range [0, N-1].
0015 %
0016 % VERSION:    $Id: upsample.m,v 1.2 2007/11/26 14:48:13 ingo Exp $
0017 %
0018 % The following call returns a parameter list object that contains the
0019 % default parameter values:
0020 %
0021 % >> pl = upsample(ao, 'Params')
0022 %
0023 % HISTORY: 19-03-07 M Hewitson
0024 %           Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0027
0028 VERSION    = '$Id: upsample.m,v 1.2 2007/11/26 14:48:13 ingo Exp $';
0029 ALGONAME    = mfilename;
0030 bs          = [];
0031
0032 %% Check if this is a call for parameters
0033 if nargin == 2
0034     if isa(varargin{1}, 'ao') && ischar(varargin{2})

```

```

0035     in = char(varargin{2});
0036     if strcmp(in, 'Params')
0037         bs = getDefaultPL();
0038         return
0039     elseif strcmp(in, 'Version')
0040         bs = VERSION;
0041         return
0042     end
0043 end
0044 end
0045
0046 VERSION = '$Id: upsample.m,v 1.2 2007/11/26 14:48:13 ingo Exp $';
0047
0048 %% capture input variables
0049 invars = {};
0050 as      = [];
0051 pl      = [];
0052
0053 for j=1:nargin
0054     a = varargin{j};
0055     if isa(a, 'ao')
0056         d = a.data;
0057         if isa(d, 'tsdata')
0058             as = [as a];
0059
0060             ao_name = inputname(j);
0061             if isempty(ao_name)
0062                 ao_name = 'no_ao_name';
0063             end
0064
0065             % Memorise the variable name of the corresponding analysis object.
0066             % If the ao is an array or vector add the index to the variable name
0067             if numel(varargin{j}) == 1
0068                 invars{end+1} = ao_name;
0069             else
0070                 for ii=1:numel(varargin{j})
0071                     [I,J] = ind2sub(size(varargin{j}),ii);
0072                     invars{end+1} = sprintf('%s(%d,%d)', ao_name, I, J);
0073                 end
0074             end
0075         else
0076             warning('!!! ignoring non-time-series AO. ');
0077         end
0078     end
0079     if isa(a, 'plist')
0080         pl = a;
0081     end
0082 end
0083
0084
0085 if isempty(pl)
0086     error('### Please give a plist with a parameter ''N''.');
0087 end

```

```

0088
0089 % Get output sample rate
0090 Nup = find(pl, 'N');
0091 if isempty(Nup)
0092     error('### Please give a plist with a parameter ''N''.');
0093 end
0094 % Get initial phase
0095 phase = find(pl, 'phase');
0096 if isempty(phase)
0097     phase = 0;
0098 end
0099
0100 for j=1:numel(as)
0101     d = as(j).data;
0102
0103     % upsample y
0104     y = upsample(d.y, floor(Nup), phase);
0105
0106     % make new output tsdata
0107     do = tsdata(y, Nup*d.fs);
0108     d = set(d, 'x', do.x);
0109     d = set(d, 'y', do.y);
0110     d = set(d, 'fs', do.fs);
0111
0112     %----- create output AO
0113
0114     % make a new history object
0115     h = history(ALGONAME, VERSION, pl, as(j).hist);
0116     h = set(h, 'invars', invars);
0117
0118     % make output analysis object
0119     b = ao(d, h);
0120     b = setnh(b, 'name', sprintf('upsample(%s)', invars{j}));
0121
0122     bs = [bs b];
0123
0124
0125 end
0126
0127 % Reshape the output to the same size of the input
0128 bs = reshape(bs, size(as));
0129
0130 % Get default params
0131 function plo = getDefaultPL()
0132
0133 plo = plist();
0134 plo = append(plo, param('N', 1));
0135 plo = append(plo, param('phase', 0));
0136
0137 % END

```

Generated on Tue 22-Jan-2008 10:39:13 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

%%%

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 pl_default = getDefaultPL()**

SOURCE CODE

```
0001 function ao_out = var (varargin)
0002 % VAR overloads the var operator for Analysis objects. Compute the variance.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: VAR overloads the var operator for Analysis objects.
0007 %               Compute the variance.
0008 %               VAR(ao) is the variance of the values in ao.data.
0009 %
0010 % CALL: ao_out = var(ao_in);
0011 %       ao_out = var(ao_in, dim);
0012 %       ao_out = var(ao_in, W, DIM); --> use the parameterlist
0013 %       ao_out = var(ao_in, pl);
0014 %       ao_out = var(ao1, pl1, ao_vector, ao_matrix, pl2);
0015 %
0016 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0017 %                   ao_in  = ao_vector
0018 %                   ao_in  = ao_matrix
0019 %
0020 % PARAMETER LIST:  <key>          <value>          <description>
0021 %                  'w'            []                represent the weight
vector W
0022 %                  'dim'          1 or 2 or 3 ...    takes the var along the
0023 %                                                              dimension dim
0024 %                  tsdata  fsdata  xydata
0025 %                  'xdata'  't'     'f'     'x'      compute the var of the x-
axis
0026 %                  'ydata'  'x'     'xx'    'y'      compute the var of the y-
axis
```

```

0027 %
0028 % The following call returns a parameter list object that contains the
0029 % default parameter values:
0030 %
0031 % >> pl = var(ao, 'Params')
0032 %
0033 % HISTORY: 23-05-2007 Diepholz
0034 %           Creation
0035 %
0036 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0037
0038 VERSION = '$Id: var.m,v 1.8 2007/11/26 14:48:13 ingo Exp $';
0039 ao_out   = [];
0040 pl       = plist();
0041
0042 %% Check if this is a call for parameters
0043 if nargin == 2
0044     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0045         in = char(varargin{2});
0046         if strcmp(in, 'Params')
0047             ao_out = getDefaultPL();
0048             return
0049         elseif strcmp(in, 'Version')
0050             ao_out = VERSION;
0051             return
0052         end
0053     end
0054 end
0055
0056 %% Collect input ao's, plist's and ao variable names
0057 in_names = {};
0058 ps       = plist();
0059 for ii = 1:nargin
0060     in_names{end+1} = inputname(ii);
0061
0062     if isnumeric(varargin{ii})
0063         ps = [ps plist(param('dim', varargin{ii}))];
0064     end
0065 end
0066
0067 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0068
0069 pl = combine(pl, ps);
0070
0071 %% go through analysis objects
0072 for j=1:numel(ao_set)
0073     a = ao_set(j);
0074
0075     [h, var_data] = single_operation(a.data, 'var', pl);
0076
0077     %% Add the history from the ao object to the history
0078     h = set(h, 'inhists', [a.hist]);

```

```

0079
0080 %% Set the var_name to the history
0081 h = set(h, 'invars', cellstr(invars{j}));
0082
0083 do_xdata = find(pl, 'xdata');
0084 do_ydata = find(pl, 'ydata');
0085
0086 %% Is no axis entry in the parameter list
0087 %% then set the default axis = xdata
0088 if isempty(do_xdata) && isempty(do_ydata)
0089     do_ydata = 'yes';
0090 end
0091
0092 %% Get the operation result. It is stored in the individual axis
0093 pl_get_axis = plist([param('xdata', do_xdata) param('ydata', do_ydata)]);
0094 [var_x, var_y] = get_xy_values(var_data, pl_get_axis);
0095
0096 var_cdata = cdata([var_x var_y]);
0097 var_cdata = set (var_cdata, 'name', var_data.name);
0098 var_cdata = set (var_cdata, 'xunits', var_data.xunits);
0099 var_cdata = set (var_cdata, 'yunits', var_data.yunits);
0100
0101 %% create a new analysis objects
0102 new_ao = ao (var_cdata, h);
0103 new_ao = setnh(new_ao, 'name', sprintf('var(%s)', invars{j}) );
0104
0105 ao_out = [ao_out new_ao];
0106
0107 end
0108
0109 % Reshape the ouput to the same size of the input
0110 ao_out = reshape(ao_out, size(ao_set));
0111
0112
0113 %% Get default params
0114 function pl_default = getDefaultPL()
0115
0116     pl_default = plist([param('dim',    '')
0117                          param('w',     [])
0118                          param('xdata',  '')
0119                          param('ydata',  '')[ ]]);
0120
0121 % END

```


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.
 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.
 xml	XML convert a plist object to an DOM XML object.
 xmladd	XMLADD add a plist object to the given node of the input XML DOM
 xmlparse	XMLPARSE creates a parameter list from a plist xml file.

Subsequent directories:

- CVS
- [private](#)

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@plist](#) > char.m

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 1.12 2007/12/11 19:31:02 ingo 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.
- [eq](#) EQ overloads the == operator for plist objects.
- [find](#) FIND overloads find routine for a parameter list.
- [get](#) GET get parameter list properties.

- **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.
- **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.
- **xmlparse** XMLPARSE creates a parameter list from a plist xml file.

SOURCE CODE

```

0001 function pstr = char(pl, varargin)
0002 % CHAR convert a parameter list into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a parameter list into a string.
0007 %
0008 % CALL:          string = char(plist);
0009 %
0010 % VERSION:       $Id: char.m,v 1.12 2007/12/11 19:31:02 ingo Exp $
0011 %
0012 % HISTORY:       04-02-2007 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: char.m,v 1.12 2007/12/11 19:31:02 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(pl, 'plist') && ischar(varargin{1})
0022         in = char(varargin{1});
0023         if strcmp(in, 'Params')
0024             pstr = plist;
0025             return
0026         elseif strcmp(in, 'Version')
0027             pstr = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 np    = length(pl.params);
0034 pstr = '';
0035 if np > 0
0036     pstr = ['('];
0037     for j=1:np

```

```

0038     p = pl.params(j);
0039
0040     %%%%%%%%%%% cell %%%%%%%%%%%
0041     if iscell(p.val)
0042         pstr = [pstr p.key '='];
0043         for ii = 1:length(p.val)
0044             if ischar(p.val{ii})
0045                 pstr = [pstr '' p.val{ii} ', '];
0046             elseif isnumeric(p.val{ii})
0047                 pstr = [pstr '' ltpda_mat2str(p.val{ii}) ', '];
0048             else
0049                 pstr = [pstr '' class(p.val{ii}) '-object, '];
0050             end
0051         end
0052         if strcmp(pstr(end-1:end), ', ')
0053             pstr = [pstr(1:end-2) '}', '];
0054         else
0055             pstr = [pstr '}', '];
0056         end
0057
0058     %%%%%%%%%%% character %%%%%%%%%%%
0059     elseif ischar(p.val)
0060         sval = char(p.val);
0061         if length(sval) == 0
0062             sval = ['' ''];
0063         elseif length(sval) >= 10
0064             sval = [sval(1:10) '...'];
0065         end
0066         pstr = [pstr p.key '=' sval ', '];
0067
0068     %%%%%%%%%%% numbers %%%%%%%%%%%
0069     elseif isnumeric(p.val)
0070         val_str = ltpda_mat2str(p.val);
0071         if length(val_str) > 25
0072             val_str = [val_str(1:25) '...'];
0073         end
0074         pstr = [pstr p.key '=' val_str ', '];
0075
0076     %%%%%%%%%%% specwin %%%%%%%%%%%
0077     elseif isa(p.val, 'specwin')
0078         % display only the window name here
0079         w = p.val;
0080         pstr = [pstr p.key '=' w.name ', '];
0081
0082     %%%%%%%%%%% miir %%%%%%%%%%%
0083     elseif isa(p.val, 'miir')
0084         f = p.val;
0085         if strcmp(get(f, 'infile'), '')
0086             pstr = [pstr p.key '=' get(f, 'name') ', '];
0087         else
0088             pstr = [pstr p.key '=' get(f, 'infile') ', '];
0089         end

```

```

0090
0091     %%%%%%%%%%    mfir    %%%%%%%%%%
0092     elseif isa(p.val, 'mfir')
0093         f = p.val;
0094         pstr = [pstr p.key '=' get(f, 'name') ', '];
0095
0096     %%%%%%%%%%    time, timeformat, timespan    %%%%%%%%%%
0097     elseif isa(p.val, 'time') || isa(p.val, 'timeformat') || isa(p.val,
'timespan')
0098         pstr = [pstr p.key '=' char(p.val) ', '];
0099
0100     %%%%%%%%%%    all other classes    %%%%%%%%%%
0101     else
0102         pstr = [pstr p.key '=' class(p.val) '-object, '];
0103 %     else
0104 %         warning('!!! unknown parameter type. Can''t convert to string.');
```

```

0105     end
0106 end
0107 % remove last ', '
0108 if length(pstr)>1
0109     pstr = pstr(1:end-2);
0110 end
0111 % close bracket
0112 pstr = [pstr ')'];
0113 end
0114
0115
0116 % END
```

[Home](#) > [classes](#) > [@plist](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a parameter list from a plist xml file.

SYNOPSIS

function ps = xmlparse(p, xmlfile)

DESCRIPTION

XMLPARSE creates a parameter list from a plist xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a parameter list into a string.

This function is called by:

- [plist](#) PLIST Plist class object constructor.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a parameter list from a plist xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```

```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'plist')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```


[Home](#) > [classes](#) > [@plist](#) > [append.m](#)

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')

REMAKR: 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 1.7 2007/12/11 19:31:02 ingo 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

```

0001 function pl = append(pl, varargin)
0002 % APPEND append a parameter to the parameter list.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: APPEND append a parameter to the parameter list.
0007 %
0008 % CALL:          pl = append(pl,p);
0009 %                pl = append(pl,pl);
0010 %                pl = append(pl, 'key1', 'value1')
0011 %
0012 % REMAKR:        It is not possible to append an key/value pair if the key exist
0013 %                in the parameter list. Tn this case an error is thrown.
0014 %
0015 % VERSION:       $Id: append.m,v 1.7 2007/12/11 19:31:02 ingo Exp $
0016 %
0017 % HISTORY:       30-01-07 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: append.m,v 1.7 2007/12/11 19:31:02 ingo Exp $';
0023
0024 % Check if this is a call for parameters
0025 if nargin == 2
0026     if isa(pl, 'plist') && ischar(varargin{1})
0027         in = char(varargin{1});
0028         if strcmp(in, 'Params')
0029             pl = plist();
0030             return
0031         elseif strcmp(in, 'Version')
0032             pl = VERSION;
0033             return
0034         end
0035     end
0036 end
0037
0038 argin = varargin;
0039
0040 while ~isempty(argin)
0041
0042     if isa(argin{1}, 'param')
0043         pp = argin{1};
0044         pl = add_param(pl, pp);

```

```

0045     argin = argin(2:end);
0046
0047     elseif isa(argin{1}, 'plist')
0048         pl_in = argin{1};
0049
0050         for ii = 1:nparams(pl_in)
0051             pl = add_param(pl, pl_in.params(ii));
0052         end
0053         argin = argin(2:end);
0054
0055     else
0056         if length(argin) >= 2
0057             key = argin{1};
0058             val = argin{2};
0059             pl = add_param(pl, param(key, val));
0060             argin = argin(3:end);
0061         else
0062             error('### There is a key [%s] without a value left.', key);
0063             argin = argin(2:end);
0064         end
0065     end
0066 end
0067
0068
0069 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0070 %
0071 % DESCRIPTION: The input parameter will only be added if the key doesn't exist
0072 %               in the plist. Throw an error if the key exist in the plist.
0073 %
0074 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0075 function pl = add_param(pl, param)
0076
0077 for ii = 1:nparams(pl)
0078     if strcmpi(pl.params(ii).key, param.key)
0079         error('### The key [%s] exist in the parameter list.\n\n### Please use the
function pset.', param.key);
0080         % pl.params(ii).val = param.val;
0081         return
0082     end
0083 end
0084
0085 % Convert all key characters into upper case characters
0086 param.key = upper(param.key);
0087
0088 pl.params = [pl.params param];
0089
0090 % END
0091

```

[Home](#) > [classes](#) > [@plist](#) > combine.m

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 1.7 2007/12/11 19:31:02 ingo 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:

SUBFUNCTIONS

- function pl = add_param(pl, param)

SOURCE CODE

```

0001 function plo = combine(varargin)
0002 % COMBINE multiple parameter lists (plist objects) into a single plist.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: COMBINE multiple parameter lists (plist objects) into a single
0007 %               plist. Duplicate parameters are given priority in the order
0008 %               in which they appear in the input.
0009 %
0010 % CALL:        pl = combine(p1, p2, p3);
0011 %              pl = combine(p1, [p2 p3], p4)
0012 %
0013 % EXAMPLES:    >> pl1 = plist(param('A', 1));
0014 %              >> pl2 = plist(param('A', 3));
0015 %              >> plo = plist(pl1, pl2)
0016 %
0017 %              Then plo will contain a parameter 'A' with value 1.
0018 %
0019 % VERSION:     $Id: combine.m,v 1.7 2007/12/11 19:31:02 ingo Exp $
0020 %
0021 % HISTORY:     28-04-2007 M Hewitson
0022 %              Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 VERSION = '$Id: combine.m,v 1.7 2007/12/11 19:31:02 ingo Exp $';
0027
0028 % Check if this is a call for parameters
0029 if nargin == 2
0030     if isa(varargin{1}, 'plist') && ischar(varargin{2})
0031         in = char(varargin{2});
0032         if strcmp(in, 'Params')
0033             plo = plist;
0034             return
0035         elseif strcmp(in, 'Version')
0036             plo = VERSION;
0037             return
0038         end
0039     end
0040 end
0041
0042 % get input plists
0043 % this code allows for inputs of the form:

```

```

0044 % >> pl = combine([p1 p2], p, [p3 p4], p5, p6)
0045 % or some combination of that.
0046 pls = [];
0047 for j=1:length(varargin)
0048     pj = varargin{j};
0049     if isa(pj, 'plist')
0050         pj = reshape(pj, 1, []);
0051         pls = [pls pj];
0052     end
0053 end
0054
0055 % The properties of the output plist is corresponding to the input plist
0056 plo = pls(1);
0057
0058 % Loop over all plist's
0059 for ii = 2:length(pls)
0060
0061     % Loop over all params in the current plist
0062     for jj = 1:length(pls(ii).params)
0063
0064         ps = pls(ii).params(jj);
0065         plo = add_param(plo, ps);
0066
0067     end
0068
0069 end
0070
0071
0072 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0073 %
0074 % DESCRIPTION: The input parameter will only be added if the key doesn't exist
0075 %               in the plist.
0076 %
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078 function pl = add_param(pl, param)
0079
0080 for ii = 1:nparams(pl)
0081     if strcmpi(pl.params(ii).key, param.key)
0082         % If the key exist in pl then do nothing, otherwise add the key value pair.
0083         return
0084     end
0085 end
0086
0087 % Convert all key characters into upper case characters
0088 param.key = upper(param.key);
0089
0090 pl.params = [pl.params param];
0091
0092
0093 % END
0094

```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@plist](#) > display.m

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 1.4 2007/12/12 18:45:08 ingo 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.

SUBFUNCTIONS

- [function](#) new_txt = [single_cell](#)(txt_field)

SOURCE CODE

```

0001 function txt = display(pls, varargin)
0002 % DISPLAY display plist object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY display plist object.
0007 %
0008 % VERSION: $Id: display.m,v 1.4 2007/12/12 18:45:08 ingo Exp $
0009 %
0010 % HISTORY: 30-01-2007 M Hewitson
0011 %             Creation
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014
0015 VERSION = '$Id: display.m,v 1.4 2007/12/12 18:45:08 ingo Exp $';
0016
0017 % Check if this is a call for parameters
0018 if nargin == 2
0019     if isa(pls, 'plist') && ischar(varargin{1})
0020         in = char(varargin{1});
0021         if strcmp(in, 'Params')
0022             txt = plist();
0023             return
0024         elseif strcmp(in, 'Version')
0025             txt = VERSION;
0026             return
0027         end
0028     end
0029 end
0030
0031 txt = {};
0032
0033 for ii=1:numel(pls)
0034     pl = pls(ii);
0035     n = nparams(pl);
0036
0037     banner_start = sprintf('----- plist %02d -----', ii);
0038     txt{end+1} = banner_start;
0039
0040     txt{end+1} = sprintf('n params: %d', n);
0041     params = get(pl, 'params');
0042     if n>0
0043         txt{end+1} = display(params);
0044     end
0045
0046     txt = single_cell(txt);
0047
0048     banner_end(1:length(banner_start)) = '-';
0049     txt{end+1} = banner_end;

```

```
0050
0051 end
0052
0053 if nargout == 0
0054     for ii=1:length(txt)
0055         disp(txt{ii});
0056     end
0057 end
0058
0059
0060 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0061
0062 function new_txt = single_cell(txt_field)
0063
0064 new_txt = {};
0065 for ii=1:length(txt_field)
0066     if iscell(txt_field{ii})
0067         hh = single_cell(txt_field{ii});
0068         new_txt(end+1:end+length(hh)) = hh(1:end);
0069     else
0070         new_txt{end+1} = txt_field{ii};
0071     end
0072 end
0073
0074 % 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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(c1,c2, 'ao/name')

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 1.7 2007/12/12 18:45:08 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:

- **eq** EQ overloads the == operator for plist objects.
- **ne** NE overloads the ~= operator for param objects.

SOURCE CODE

```

0001 function result = eq(c1, c2, varargin)
0002 % EQ overloads the == operator for plist objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EQ overloads the == operator for plist objects.
0007 %
0008 %           All fields are checked.
0009 %
0010 % CALL:      result = eq(c1,c2)
0011 %           result = eq(c1,c2,  exc_list)
0012 %           result = eq(c1,c2, 'property1', 'property2')
0013 %           result = eq(c1,c2, 'class/property', 'class/property')
0014 %
0015 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0016 %           result = eq(c1,c2, 'ao/name')
0017 %
0018 % INPUTS:    c1,c2      - input plist objects
0019 %           exc_list - exception list
0020 %           List of properties which are not checked.
0021 %
0022 % OUTPUTS:   If the two plist objects are considered equal, result == 1,
0023 %           otherwise, result == 0.
0024 %
0025 % VERSION:   $Id: eq.m,v 1.7 2007/12/12 18:45:08 ingo Exp $
0026 %
0027 % HISTORY:   29-08-2007 M Hewitson
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION = '$Id: eq.m,v 1.7 2007/12/12 18:45:08 ingo Exp $';
0033
0034 % Check if this is a call for parameters
0035 if nargin == 2
0036     if isa(c1, 'plist') && ischar(c2)
0037         in = char(c2);
0038         if strcmp(in, 'Params')
0039             result = plist;

```

```

0040     return
0041     elseif strcmp(in, 'Version')
0042         result = VERSION;
0043         return
0044     end
0045 end
0046 end
0047
0048 result = 1;
0049
0050 %% Check class
0051 if ~strcmp(class(c1), class(c2))
0052     result = 0;
0053     return
0054 end
0055
0056 fields = fieldnames(c1);
0057
0058 for ii = 1:length(fields)
0059     field = fields{ii};
0060
0061     ck_field = {field, ['plist/' field]};
0062
0063     %% Check tag
0064     if ~(any(ismember(ck_field, varargin)))
0065
0066         if isobject(c1.(field))
0067
0068             if length(c1.(field)) ~= length(c2.(field))
0069                 result = 0;
0070                 return
0071             end
0072
0073             for jj = 1:length(c1.(field))
0074                 if ~eq(c1.(field)(jj), c2.(field)(jj), varargin{:})
0075                     result = 0;
0076                     disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0077                     return
0078                 end
0079             end
0080         else
0081             if ~isequalwithhequalnans(c1.(field), c2.(field))
0082                 result = 0;
0083                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0084                 return
0085             end
0086         end
0087     end
0088 end
0089 end
0090

```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@plist](#) > find.m

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 1.5 2007/12/12 18:45:08 ingo 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:

- [pzmresp](#) PZMRESP shadow function for the pzmodel/resp.
- [resp](#) RESP shadows miir/iirResp.

SOURCE CODE

```

0001 function a = find(pl, key)
0002 % FIND overloads find routine for a parameter list.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: FIND overloads find routine for a parameter list.
0007 %               Returns the value corresponding to the first parameters in
0008 %               the list with key 'key'.
0009 %
0010 % CALL:         a = find(pl, 'key')
0011 %
0012 % VERSION:      $Id: find.m,v 1.5 2007/12/12 18:45:08 ingo Exp $
0013 %
0014 % HISTORY:      02-02-2007 M Hewitson
0015 %               Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 VERSION = '$Id: find.m,v 1.5 2007/12/12 18:45:08 ingo Exp $';
0020
0021 % Check if this is a call for parameters
0022 if nargin == 2
0023     if isa(pl, 'plist') && ischar(key)
0024         in = char(key);
0025         if strcmp(in, 'Params')
0026             a = plist;
0027             return
0028         elseif strcmp(in, 'Version')
0029             a = VERSION;
0030             return
0031         end
0032     end
0033 end
0034
0035 ps = get(pl, 'params');
0036 np = length(ps);
0037
0038 a = [];
0039
0040 for i=1:np
0041     p = ps(i);
0042     if strcmpi(p.key, key)
0043         a = p.val;
0044         return
0045     end
0046 end
0047
0048
0049
0050 % END

```


[Home](#) > [classes](#) > [@plist](#) > get.m

get

PURPOSE

GET get parameter list properties.

SYNOPSIS

function val = get(pl, propName)

DESCRIPTION

GET get parameter list properties.

%%%

DESCRIPTION: GET get parameter properties.

```
CALL: name      = get(pl, 'name');
      params    = get(pl, 'params');
      version   = get(pl, 'version');
      created   = get(pl, 'created');
```

VERSION: \$Id: get.m,v 1.4 2007/10/15 17:16:56 ingo Exp \$

HISTORY: 30-01-07 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:

- [char](#) CHAR convert a parameter list into a string.
- [display](#) DISPLAY display plist object.
- [find](#) FIND overloads find routine for a parameter list.
- [remove](#) REMOVE remove a parameter from the parameter list.
- [xmladd](#) XMLADD add a plist object to the given node of the input XML DOM

SOURCE CODE

```

0001 function val = get(pl, propName)
0002 % GET get parameter list properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get parameter properties.
0007 %
0008 % CALL: name      = get(pl, 'name');
0009 %           params = get(pl, 'params');
0010 %           version = get(pl, 'version');
0011 %           created = get(pl, 'created');
0012 %
0013 % VERSION: $Id: get.m,v 1.4 2007/10/15 17:16:56 ingo Exp $
0014 %
0015 % HISTORY: 30-01-07 M Hewitson
0016 %           Creation
0017 %
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019
0020 VERSION = '$Id: get.m,v 1.4 2007/10/15 17:16:56 ingo Exp $';
0021
0022 % Check if this is a call for parameters or for the cvs-version number
0023 if nargin == 2
0024     if isa(pl, 'plist') && ischar(propName)
0025         in = char(propName);
0026         if strcmp(in, 'Params')
0027             val = plist();
0028             return
0029         elseif strcmp(in, 'Version')
0030             val = VERSION;
0031             return
0032         end
0033     end
0034 end
0035
0036 switch propName
0037     case 'name'
0038         val = pl.name;
0039     case 'params'
0040         val = pl.params;
0041     case 'version'
0042         val = pl.version;
0043     case 'created'
0044         val = pl.created;
0045     otherwise
0046         error([propName, ' is not a valid plist property.']);
0047 end

```


[Home](#) > [classes](#) > [@plist](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.3 2008/01/02 17:58:51 ingo 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

```

0001 function varargout = isfield(varargin)
0002 % ISFIELD tests if the given field is one of the object properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0007 %               properties.
0008 %
0009 % CALL:         isfield(a, 'field');
0010 %
0011 %               The following call returns a parameter list object that
0012 %               contains the default parameter values:
0013 %               >> pl = isfield(plist, 'Params')
0014 %
0015 %               The following call returns the cvs version number:
0016 %               >> ver = isfield(plist, 'Version')
0017 %
0018 % VERSION:      $Id: isfield.m,v 1.3 2008/01/02 17:58:51 ingo Exp $
0019 %
0020 % HISTORY: 25-09-07 M Hewitson
0021 %           Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: isfield.m,v 1.3 2008/01/02 17:58:51 ingo Exp $';
0026
0027 % 'Params' Call
0028 if nargin == 2
0029     if isa(varargin{1}, 'plist') && strcmp(varargin{2}, 'Params')
0030         varargout{1} = plist();
0031         return
0032     elseif isa(varargin{1}, 'plist') && strcmp(varargin{2}, 'Version')
0033         varargout{1} = VERSION;
0034         return
0035     end
0036 end
0037
0038 as = [];
0039 field = '';
0040 for j=1:nargin
0041     if isa(varargin{j}, 'plist')
0042         as = [as varargin{j}];
0043     end
0044     if ischar(varargin{j})
0045         field = varargin{j};
0046     end
0047 end
0048
0049 res = zeros(size(as));
0050 for j=1: numel(as)
0051     try
0052         t = as(j).(field);

```

```
0053     res(j) = 1;
0054     catch
0055         res(j) = 0;
0056     end
0057 end
0058
0059 varargout{1} = res;
0060
```

Generated on Tue 22-Jan-2008 10:39:13 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.

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

DESCRIPTION: ISPARAM look for a given key in the parameter list. Exist the key
              in the parameter list then is the result 1 otherwise 0.

CALL:        res = isparam(pl, 'key')

VERSION:     $Id: isparam.m,v 1.2 2007/11/23 15:34:47 ingo Exp $

HISTORY:     07-11-2007 Diepholz
              Creation

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

CROSS-REFERENCE INFORMATION

This function calls:

- [plist](#) PLIST Plist class object constructor.

This function is called by:

SOURCE CODE

```
0001 function res = isparam(pl, key)
0002 % ISPARAM look for a given key in the parameter list.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ISPARAM look for a given key in the parameter list. Exist the key
0007 %               in the parameter list then is the result 1 otherwise 0.
```



```

0008 %
0009 % CALL:          res = isparam(pl, 'key')
0010 %
0011 % VERSION:       $Id: isparam.m,v 1.2 2007/11/23 15:34:47 ingo Exp $
0012 %
0013 % HISTORY:       07-11-2007 Diepholz
0014 %                  Creation
0015 %
0016 %%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: isparam.m,v 1.2 2007/11/23 15:34:47 ingo Exp $';
0019
0020 % Check if this is a call for parameters or for the cvs-version number
0021 if nargin == 2
0022     if isa(pl, 'plist') && ischar(key)
0023         in = key;
0024         if strcmp(in, 'Params')
0025             res = plist();
0026             return
0027         elseif strcmp(in, 'Version')
0028             res = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 p    = pl.params;
0035 res = 0;
0036
0037 for ii = 1:length(p)
0038     if strcmpi(p(ii).key, key)
0039         res = 1;
0040         return
0041     end
0042 end
0043
0044
0045
0046 % END

```

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')

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 1.3 2008/01/03 15:33:48 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

```

0001 function result = ne(p1, p2, varargin)
0002 % NE overloads the ~= operator for param objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for param objects.
0007 %
0008 % CALL:          result = ne(p1,p2)
0009 %               result = ne(p1,p2, 'property1', 'property2')
0010 %               result = ne(p1,p2, 'class/property', 'class/property')
0011 %
0012 % INPUTS:        p1,p2 - input param objects
0013 %
0014 % OUTPUTS:       If the two param objects are considered equal, result == 0,
0015 %               otherwise, result == 1.
0016 %
0017 % VERSION:       $Id: ne.m,v 1.3 2008/01/03 15:33:48 ingo Exp $
0018 %
0019 % HISTORY:       29-08-2007 M Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: ne.m,v 1.3 2008/01/03 15:33:48 ingo Exp $';
0025
0026 % Check if this is a call for parameters
0027 if nargin == 2
0028     if isa(p1, 'plist') && ischar(p2)
0029         in = char(p2);
0030         if strcmp(in, 'Params')
0031             result = plist;
0032             return
0033         elseif strcmp(in, 'Version')
0034             result = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 if eq(p1, p2, varargin{:})
0041     result = 0;
0042 else
0043     result = 1;
0044 end
0045
0046 % END

```

[Home](#) > [classes](#) > [@plist](#) > nparams.m

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 1.3 2007/12/12 18:45:09 ingo 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

```

0001 function n = nparams(pl, varargin)
0002 % NPARAMS returns the number of param objects in the list.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NPARAMS returns the number of param objects in the list.
0007 %
0008 % CALL:          n = nparams(plist)
0009 %
0010 % VERSION:       $Id: nparams.m,v 1.3 2007/12/12 18:45:09 ingo Exp $
0011 %
0012 % HISTORY:       30-01-2007 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: nparams.m,v 1.3 2007/12/12 18:45:09 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(pl, 'plist') && ischar(varargin{1})
0022         in = char(varargin{1});
0023         if strcmp(in, 'Params')
0024             n = plist;
0025             return
0026         elseif strcmp(in, 'Version')
0027             n = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 n = length(pl.params);
0034
0035 % END

```

[Home](#) > [classes](#) > [@plist](#) > pset.m

pset

PURPOSE

PSET set parameter list properties.

SYNOPSIS

function pl = pset(pl, varargin)

DESCRIPTION

PSET set parameter list properties.

%%%

DESCRIPTION: PSET set or add a key/value pair into the parameter list. Exist
the key in the parameter list then becomes the value the new
value.

CALL: pl = set(pl, key, val);
 pl = set(pl, key1, val1, key2, val2);

VERSION: \$Id: pset.m,v 1.4 2007/11/23 15:34:47 ingo Exp \$

HISTORY: 30-01-07 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:

SOURCE CODE

```
0001 function pl = pset(pl, varargin)
0002 % PSET set parameter list properties.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: PSET set or add a key/value pair into the parameter list. Exist
0007 %               the key in the parameter list then becomes the value the new
0008 %               value.
0009 %
0010 % CALL:         pl = set(pl, key, val);
0011 %               pl = set(pl, key1, val1, key2, val2);
0012 %
0013 % VERSION:      $Id: pset.m,v 1.4 2007/11/23 15:34:47 ingo Exp $
0014 %
0015 % HISTORY:      30-01-07 M Hewitson
0016 %               Creation
0017 %
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019
0020 VERSION = '$Id: pset.m,v 1.4 2007/11/23 15:34:47 ingo Exp $';
0021
0022 % Check if this is a call for parameters or for the cvs-version number
0023 if nargin == 2
0024     if isa(pl, 'plist') && ischar(varargin{1})
0025         in = varargin{1};
0026         if strcmp(in, 'Params')
0027             pl = plist();
0028             return
0029         elseif strcmp(in, 'Version')
0030             pl = VERSION;
0031             return
0032         end
0033     end
0034 end
0035
0036 if (nargout ~= 0)
0037
0038     propArgin = varargin;
0039
0040     while length(propArgin) >= 1
0041
0042         %%%% Set a parameter object %%%%
0043         if isa(propArgin{1}, 'param')
0044             prop = upper(propArgin{1}.key);
0045             val = propArgin{1}.val;
0046             propArgin = propArgin(2:end);
0047
0048             %%%% Set key/value pair %%%%
0049         elseif length(propArgin) >= 2
0050             prop = upper(propArgin{1});
0051             val = propArgin{2};
0052             propArgin = propArgin(3:end);
0053
0054         else
0055             error('### There is only one key[%s] without a value left.',char(propArgin
{1}));

```

```

0056     end
0057
0058     % check to see if this is one of the parameters we can set, otherwise
0059     % add it
0060     found = 0;
0061     for j=1:length(pl.params)
0062         if strcmpi(pl.params(j).key, prop)
0063             pl.params(j).key = prop;
0064             pl.params(j).val = val;
0065             found = 1;
0066         end
0067     end
0068     % add this parameter if necessary
0069     if ~found
0070         pl = append(pl, param(prop, val));
0071     end
0072
0073 end
0074
0075 else
0076     if ischar(varargin{2})
0077         error('### please use: %s = set(%s, '%s'', '%s'');', ...
0078             inputname(1), ...
0079             inputname(1), ...
0080             varargin{1}, ...
0081             varargin{2});
0082     elseif isnumeric(varargin{2})
0083         error('### please use: %s = set(%s, '%s'', %d);', ...
0084             inputname(1), ...
0085             inputname(1), ...
0086             varargin{1}, ...
0087             varargin{2});
0088     else
0089         error('### please use: %s = set(%s, key, 'value'');', ...
0090             inputname(1), ...
0091             inputname(1));
0092     end
0093 end
0094
0095 % END

```


[Home](#) > [classes](#) > [@plist](#) > pzmpresp.m

pzmpresp

PURPOSE [^](#)

PZMRESP shadow function for the pzmodel/resp.

SYNOPSIS [^](#)

function a = pzmpresp(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 = pzmpresp(pl)

VERSION: \$Id: pzmpresp.m,v 1.3 2007/12/12 18:45:09 ingo 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 [^](#)

```
0001 function a = pzmpresp(pl, varargin)
0002 % PZMRESP shadow function for the pzmodel/resp.
```

```

0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: PZMRESP shadow function for the pzmodel/resp.
0007 %             A parameter called 'pzmodel' is extracted and passed with the
0008 %             remaining parameter list to pzmodel/resp.
0009 %
0010 % CALL:       resp_pzm = pzmpresp(pl)
0011 %
0012 % VERSION:    $Id: pzmpresp.m,v 1.3 2007/12/12 18:45:09 ingo Exp $
0013 %
0014 % HISTORY:    19-04-2007 M Hewitson
0015 %             Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 VERSION = '$Id: pzmpresp.m,v 1.3 2007/12/12 18:45:09 ingo Exp $';
0020
0021 %%%% Check if this is a call for parameters %%%%
0022 if nargin == 2
0023     if isa(pl, 'plist') && ischar(varargin{1})
0024         in = char(varargin{1});
0025         if strcmp(in, 'Params')
0026             a = plist;
0027             return
0028         elseif strcmp(in, 'Version')
0029             a = VERSION;
0030             return
0031         end
0032     end
0033 end
0034
0035 pzm = find(pl, 'pzmodel');
0036 if isempty(pzm) || ~isa(pzm, 'pzmodel')
0037     error('### no pzmodel found in parameter list.')
0038 end
0039
0040 if nargout == 0
0041     resp(pzm, pl);
0042 else
0043     a = resp(pzm, pl);
0044 end
0045
0046 % END

```

[Home](#) > [classes](#) > [@plist](#) > remove.m

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 1.4 2008/01/02 17:58:51 ingo 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:

SOURCE CODE

```
0001 function pl = remove(pl, val)
0002 % REMOVE remove a parameter from the parameter list.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

0005 %
0006 % DESCRIPTION: REMOVE remove a parameter from the parameter list.
0007 %
0008 % CALL:          pl = remove(pl, 2)          - removes 2nd in the list
0009 %                pl = remove(pl, 'key')      - removes parameter called 'key'
0010 %
0011 % VERSION:       $Id: remove.m,v 1.4 2008/01/02 17:58:51 ingo Exp $
0012 %
0013 % HISTORY:       30-01-2007 M Hewitson
0014 %                Creation
0015 %
0016 %%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: remove.m,v 1.4 2008/01/02 17:58:51 ingo Exp $';
0019
0020 %%%% 'Params' && 'Version' Call %%%%
0021 if nargin == 2
0022     if isa(pl, 'plist') && strcmp(val, 'Params')
0023         pl = plist();
0024         return
0025     elseif isa(pl, 'plist') && strcmp(val, 'Version')
0026         pl = VERSION;
0027         return
0028     end
0029 end
0030
0031 np = nparams(pl);
0032 params = get(pl, 'params');
0033
0034 %%%% Remove specified key %%%%
0035 if ischar(val)
0036     idx = [];
0037     for i=1:np
0038         key = get(params(i), 'key');
0039         if ~strcmpi(key, val)
0040             idx = [idx i];
0041         end
0042     end
0043     pl = plist(params(idx));
0044     if length(idx) == np
0045         warning('!!! No parameter found matching key name');
0046     end
0047
0048 %%%% Remove specified position %%%%
0049 elseif isnumeric(val)
0050
0051     idx = [1:val-1 val+1:np];
0052     if max(idx)>np
0053         error('### index exceeds number of parameters');
0054     end
0055     pl = plist(params(idx));

```

```
0056
0057 %%%%      Unknown method      %%%%
0058 else
0059     error('### unknown indexing method')
0060 end
0061
0062
0063
0064 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@plist](#) > resp.m

resp

PURPOSE

RESP shadows miir/iirResp.

SYNOPSIS

function [varargout] = resp(varargin)

DESCRIPTION

RESP shadows miir/iirResp.

%%%

DESCRIPTION: RESP shadows miir/iirResp.
The parameter with key 'filter' is extracted and
used to call miir/resp.

CALL: resp(plist);
 filt_resp = resp(plist);

VERSION: \$Id: resp.m,v 1.3 2008/01/02 17:58:51 ingo Exp \$

HISTORY: 12-04-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [find](#) FIND overloads find routine for a parameter list.
- [plist](#) PLIST Plist class object constructor.
- [resp](#) RESP shadows miir/iirResp.

This function is called by:

- [pzmresp](#) PZMRESP shadow function for the pzmodel/resp.
- [resp](#) RESP shadows miir/iirResp.

SOURCE CODE

```

0001 function [varargout] = resp(varargin)
0002 % RESP shadows miir/iirResp.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: RESP shadows miir/iirResp.
0007 %               The parameter with key 'filter' is extracted and
0008 %               used to call miir/resp.
0009 %
0010 % CALL:          resp(plist);
0011 %               filt_resp = resp(plist);
0012 %
0013 % VERSION:      $Id: resp.m,v 1.3 2008/01/02 17:58:51 ingo Exp $
0014 %
0015 % HISTORY:      12-04-2007 M Hewitson
0016 %               Creation
0017 %
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019
0020 VERSION = '$Id: resp.m,v 1.3 2008/01/02 17:58:51 ingo Exp $';
0021
0022 %%%% 'Params' && 'Version' Call %%%%
0023 if nargin == 2
0024     if isa(varargin{1}, 'plist') && strcmp(varargin{2}, 'Params')
0025         varargout{1} = plist();
0026         return
0027     elseif isa(varargin{1}, 'plist') && strcmp(varargin{2}, 'Version')
0028         varargout{1} = VERSION;
0029         return
0030     end
0031 end
0032
0033
0034 pl = varargin{1};
0035 if ~isa(pl, 'plist')
0036     error('### iirResp needs an input plist.')
0037 end
0038
0039 filt = find(pl, 'filter');
0040 if isempty(filt)
0041     error('### input parameter list must contain a ''filter'' parameter.');
```

```
0052 if nargout == 1
0053     varargout{1} = b;
0054 end
0055 if nargout > 1
0056     error('incorrect output arguments');
0057 end
0058
0059
0060
0061
0062
0063 % END
```


[Home](#) > [classes](#) > [@plist](#) > save.m

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 1.1 2007/08/29 18:38:13 hewitson Exp \$

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

```
>> pl = save(plist, 'Params')
```

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE a plist object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for plist objects.
0008 %
0009 % CALL:          Save a plist object as a .mat file.
0010 %                >> save(p, 'blah.mat')
0011 %                Save a plist object as an XML file.
0012 %                >> save(p, 'blah.xml')
0013 %                Save a plist object to file specified in plist.
0014 %                >> save(p, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:      $Id: save.m,v 1.1 2007/08/29 18:38:13 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(plist, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'plist') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040

```

```
0041 ALGONAME = mfilename;
0042 VERSION   = '$Id: save.m,v 1.1 2007/08/29 18:38:13 hewitson Exp $';
0043
0044 if nargin ~= 2
0045     help mfilename;
0046     error('### incorrect inputs');
0047 end
0048
0049 % get object
0050 obj = varargin{1};
0051 if isa(varargin{2}, 'plist')
0052     pl = varargin{2};
0053 elseif ischar(varargin{2})
0054     pl = plist(param('filename', varargin{2}));
0055 else
0056     help mfilename;
0057     error('### incorrect inputs');
0058 end
0059
0060 % Save object
0061 ltpda_saveobj(obj, pl);
0062
0063 % Get default params
0064 function plo = getDefaultPL()
0065
0066 disp('* creating default plist...');
0067 plo = plist();
0068 plo = append(plo, param('filename', 'object.xml'));
0069 disp('* done. ');
0070
0071
0072 % END
```

[Home](#) > [classes](#) > [@plist](#) > set.m

set

PURPOSE

SET set parameter list properties.

SYNOPSIS

function pl = set(pl, varargin)

DESCRIPTION

SET set parameter list properties.

%%%

DESCRIPTION: SET set parameter list properties.

CALL: pl = set(pl, 'name', 'my name');
 pl = set(pl, 'params', p);
 pl = set(pl, 'version', 'plist/set.m,v 1.7...');
 pl = set(pl, 'created', time-object);

REMARK: To add a new key/value pair use the function: pset.

VERSION: \$Id: set.m,v 1.11 2008/01/02 17:58:51 ingo Exp \$

HISTORY: 30-01-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [plist](#) PLIST Plist class object constructor.

This function is called by:

SOURCE CODE

```
0001 function pl = set(pl, varargin)
0002 % SET set parameter list properties.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET set parameter list properties.
0007 %
0008 % CALL:          pl = set(pl, 'name',    'my name');
0009 %                pl = set(pl, 'params',  p);
0010 %                pl = set(pl, 'version', 'plist/set.m,v 1.7...');
0011 %                pl = set(pl, 'created',  time-object);
0012 %
0013 % REMARK:        To add a new key/value pair use the function: pset.
0014 %
0015 % VERSION:       $Id: set.m,v 1.11 2008/01/02 17:58:51 ingo Exp $
0016 %
0017 % HISTORY:       30-01-07 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: set.m,v 1.11 2008/01/02 17:58:51 ingo Exp $';
0023
0024 %%%% 'Params' && 'Version' Call %%%%
0025 if nargin == 2
0026     if isa(pl, 'plist') && ischar(varargin{1})
0027         in = varargin{1};
0028         if strcmp(in, 'Params')
0029             pl = plist();
0030             return
0031         elseif strcmp(in, 'Version')
0032             pl = VERSION;
0033             return
0034         end
0035     end
0036 end
0037
0038 if (nargout ~= 0)
0039
0040     propArgin = varargin;
0041
0042     while length(propArgin) >= 2
0043         prop = propArgin{1};
0044         val  = propArgin{2};
0045         propArgin = propArgin(3:end);
0046         switch prop
0047             case 'params'
0048                 pl.params = val;
0049             case 'version'
0050                 pl.version = val;
0051             case 'name'
0052                 pl.name = val;
0053             case 'created'
0054                 if isa(val, 'time')
0055                     pl.created = val;
0056                 else

```

```
0057         error('### The property ''created'' must be a time-object');
0058     end
0059     otherwise
0060         error('### The [%s] is not a property of the plist class.\n### To set a
key/value pair use the function pset.', prop);
0061
0062     end
0063 end
0064
0065 else
0066     if ischar(varargin{2})
0067         error('### please use: %s = set(%s, ''%s'', ''%s'');', ...
0068             inputname(1), ...
0069             inputname(1), ...
0070             varargin{1}, ...
0071             varargin{2});
0072     elseif isnumeric(varargin{2})
0073         error('### please use: %s = set(%s, ''%s'', %d);', ...
0074             inputname(1), ...
0075             inputname(1), ...
0076             varargin{1}, ...
0077             varargin{2});
0078     else
0079         error('### please use: %s = set(%s, key, ''value'');', ...
0080             inputname(1), ...
0081             inputname(1));
0082     end
0083 end
0084
0085 % END
```

[Home](#) > [classes](#) > [@plist](#) > string.m

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 1.7 2008/01/02 17:58:51 ingo Exp \$

HISTORY: 29-03-2007 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a parameter list into a string.
- [plist](#) PLIST Plist class object constructor.
- [string](#) STRING converts a plist object to a command string which will recreate the plist object.

This function is called by:

- [string](#) STRING converts a plist object to a command string which will recreate the plist object.

SOURCE CODE

```
0001 function pstr = string(pl, varargin)
```

```

0002 % STRING converts a plist object to a command string which will recreate the
plist object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: STRING converts a plist object to a command string which will
0007 %             recreate the plist object.
0008 %
0009 % CALL:       cmd = string(pl)
0010 %
0011 % VERSION:    $Id: string.m,v 1.7 2008/01/02 17:58:51 ingo Exp $
0012 %
0013 % HISTORY:    29-03-2007 M Hewitson
0014 %             Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: string.m,v 1.7 2008/01/02 17:58:51 ingo Exp $';
0019
0020 %%%% 'Params' && 'Version' Call %%%%
0021 if nargin == 2
0022     if isa(pl, 'plist') && ischar(varargin{1})
0023         in = varargin{1};
0024         if strcmp(in, 'Params')
0025             pstr = plist();
0026             return
0027         elseif strcmp(in, 'Version')
0028             pstr = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 np = length(pl.params);
0035 pstr = '';
0036 if np > 0
0037     pstr = ['plist('];
0038     for j=1:np
0039         p = pl.params(j);
0040         if ischar(p.val)
0041             pstr = [pstr '' p.key '', '' char(p.val) '', '];
0042         elseif isnumeric(p.val)
0043             pstr = [pstr '' p.key '', ' ltpda_mat2str(p.val) ', '];
0044         elseif isa(p.val, 'ao')
0045             pstr = [pstr '' p.key '', ' string(p.val) ', '];
0046         elseif isa(p.val, 'history')
0047             pstr = [pstr '' p.key '', ' string(p.val) ', '];
0048         elseif isa(p.val, 'miir')
0049             pstr = [pstr '' p.key '', ' string(p.val) ', '];
0050         elseif isa(p.val, 'mfir')
0051             pstr = [pstr '' p.key '', ' string(p.val) ', '];

```



```

0052     elseif isa(p.val, 'specwin')
0053         pstr = [pstr '' p.key '', ' string(p.val) ', '];
0054     elseif isa(p.val, 'pole') || isa(p.val, 'zero') || isa(p.val, 'pzmodel')
0055         pstr = [pstr '' p.key '', ' string(p.val) ', '];
0056     elseif isa(p.val, 'time')
0057         pstr = [pstr '' p.key '', ' string(p.val) ', '];
0058     elseif iscell(p.val)
0059         val_str = ['{ '];
0060         for ii = 1:numel(p.val)
0061             cell_val = p.val{ii};
0062             if ischar(cell_val)
0063                 val_str = [val_str '' p.val{ii} '', ' ];
0064             elseif isnumeric(cell_val)
0065                 val_str = [val_str ltpda_mat2str(p.val{ii}) ', ' ];
0066             else
0067                 error('### Unknown object in cell array');
0068             end
0069         end
0070         val_str = val_str(1:end-2);
0071         val_str = [val_str ' }'];
0072         pstr = [pstr '' p.key '', ' val_str ', '];
0073     else
0074         warning('!!! unknown parameter type. Can''t convert to string. ');
0075     end
0076 end
0077
0078 % close bracket
0079 if ~isempty(pstr)
0080     pstr = pstr(1:end-2);
0081 end
0082 pstr = [pstr ')'];
0083 end
0084
0085 % END

```

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 1.6 2008/01/02 18:02:01 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for parameter list properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
```

```

0006 % DESCRIPTION: SUBSASGN define index assignment for parameter list
0007 %                  properties.
0008 %
0009 % EXAMPLES:      All possible assignment are possible.
0010 %
0011 % VERSION:       $Id: subsasgn.m,v 1.6 2008/01/02 18:02:01 ingo Exp $
0012 %
0013 % HISTORY:       31-01-07 M Hewitson
0014 %                  Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 VERSION = '$Id: subsasgn.m,v 1.6 2008/01/02 18:02:01 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(A, 'plist') && ischar(S)
0022         in = char(S);
0023         if strcmp(in, 'Params')
0024             A = plist;
0025             return
0026         elseif strcmp(in, 'Version')
0027             A = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 A = builtin ('subsasgn', A, S, B);
0034

```

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 1.13 2008/01/02 18:02:01 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for parameter list objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for parameter list objects.
```

```

0007 %
0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:        $Id: subsref.m,v 1.13 2008/01/02 18:02:01 ingo Exp $
0011 %
0012 % HISTORY:        31-01-07 M Hewitson
0013 %                  Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsref.m,v 1.13 2008/01/02 18:02:01 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'plist') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             B = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             B = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 B = builtin('subsref', A, S);
0033

```

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.
 mux	MUX concatonate 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.
 subsasgn	SUBSASGN define index assignment for parameter properties.
 subsref	SUBSREF Define field name indexing for parameter objects.
 xml	XML convert a param object to an DOM XML object.
 xmladd	XMLADD add a parameter object to the given node of the input XML
 xmlparse	XMLPARSE creates a parameter from a parameter xml file.

Subsequent directories:

- CVS
- [private](#)

[Home](#) > [classes](#) > [@param](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a parameter from a parameter xml file.

SYNOPSIS

```
function ps = xmlparse(p, xmlfile)
```

DESCRIPTION

XMLPARSE creates a parameter from a parameter xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a param object into a string.

This function is called by:

- [param](#) PARAM Parameter object class constructor.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a parameter from a parameter xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```

```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'param')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```


[Home](#) > [classes](#) > [@param](#) > save.m

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 1.1 2007/08/29 18:37:17 hewitson Exp \$

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

```
>> pl = save(param, 'Params')
```

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE a param object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for param objects.
0008 %
0009 % CALL:          Save a param object as a .mat file.
0010 %                >> save(p, 'blah.mat')
0011 %                Save a param object as an XML file.
0012 %                >> save(p, 'blah.xml')
0013 %                Save a param object to file specified in plist.
0014 %                >> save(p, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:      $Id: save.m,v 1.1 2007/08/29 18:37:17 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(param, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'param') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041
0042 ALGONAME = mfilename;

```

```
0043 VERSION = '$Id: save.m,v 1.1 2007/08/29 18:37:17 hewitson Exp $';
0044
0045 if nargin ~= 2
0046     help mfilename;
0047     error('### incorrect inputs');
0048 end
0049
0050 % get object
0051 obj = varargin{1};
0052 if isa(varargin{2}, 'plist')
0053     pl = varargin{2};
0054 elseif ischar(varargin{2})
0055     pl = plist(param('filename', varargin{2}));
0056 else
0057     help mfilename;
0058     error('### incorrect inputs');
0059 end
0060
0061 % Save object
0062 ltpda_saveobj(obj, pl);
0063
0064 % Get default params
0065 function plo = getDefaultPL()
0066
0067 disp('* creating default plist...');
0068 plo = plist();
0069 plo = append(plo, param('filename', 'object.xml'));
0070 disp('* done. ');
0071
0072
0073 % END
```

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.
 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.
 xml	XML convert a specwin object to an DOM XML object.
 xmladd	XMLADD adds specwin object to the given node of the xml input
 xmlparse	XMLPARSE creates a specwin from a specwin xml file.

Subsequent directories:

- CVS
- [private](#)

[Home](#) > [classes](#) > [@specwin](#) > char.m

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 1.3 2008/01/08 16:40:29 ingo 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.
- [eq](#) EQ overloads the == operator for specfin objects.
- [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.
- [xmlparse](#) XMLPARSE creates a specwin from a specwin xml file.

SOURCE CODE

```

0001 function pstr = char(w, varargin)
0002 % CHAR convert a specwin object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a specwin object into a string.
0007 %
0008 % CALL:          string = char(sw);
0009 %
0010 % VERSION:       $Id: char.m,v 1.3 2008/01/08 16:40:29 ingo Exp $
0011 %
0012 % HISTORY:       04-02-2007 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: char.m,v 1.3 2008/01/08 16:40:29 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(w, 'specwin') && ischar(varargin{1})
0022         in = char(varargin{1});
0023         if strcmp(in, 'Params')
0024             pstr = plist;
0025             return
0026         elseif strcmp(in, 'Version')
0027             pstr = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 pstr = [];
0034 pstr = [pstr w.name];
0035 pstr = [pstr ', alpha=' num2str(w.alpha)];
0036 pstr = [pstr ', psll=' num2str(w.psll)];
0037 pstr = [pstr ', rov=' num2str(w.rov)];
0038 pstr = [pstr ', nenbw=' num2str(w.nenbw)];
0039 pstr = [pstr ', w3db=' num2str(w.w3db)];
0040 pstr = [pstr ', flatness=' num2str(w.flatness)];
0041
0042 pstr = strrep(pstr, '_', '\_');
0043
0044 % END

```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@specwin](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a specwin from a specwin xml file.

SYNOPSIS

function ps = xmlparse(p, xmlfile)

DESCRIPTION

XMLPARSE creates a specwin from a specwin xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a specwin object into a string.

This function is called by:

- [specwin](#) SPECWIN spectral window object class constructor.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a specwin from a specwin xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```



```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'specwin')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```

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.
 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.
 xml	XML convert a pole object to an DOM XML object.
 xmladd	XMLADD add a pole object to the given node of the input XML DOM
 xmlparse	XMLPARSE creates a pole from a pole xml file.

Subsequent directories:

- CVS

- [private](#)

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@pole](#) > char.m

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 1.3 2008/01/02 17:58:51 ingo 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 concatenate poles into a vector.
- [char](#) CHAR convert a pole object into a string.
- [display](#) DISPLAY overloads display functionality for pole objects.
- [eq](#) EQ overloads the == operator for pole objects.
- [get](#) GET get pole properties.
- [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.

- [set](#) SET set a pole property.
- [subsasgn](#) SUBSASGN define index assignment for pole properties.
- [subsref](#) SUBSREF Define field name indexing for pole objects.
- [xmlparse](#) XMLPARSE creates a pole from a pole xml file.

SOURCE CODE

```

0001 function pstr = char(p, varargin)
0002 % CHAR convert a pole object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a pole object into a string.
0007 %
0008 % CALL:          string = char(pole);
0009 %
0010 % VERSION:       $Id: char.m,v 1.3 2008/01/02 17:58:51 ingo Exp $
0011 %
0012 % HISTORY:       02-04-2007 M Hewitson
0013 %                  Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: char.m,v 1.3 2008/01/02 17:58:51 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(p, 'pole') && ischar(varargin{1})
0022         in = char(varargin{1});
0023         if strcmp(in, 'Params')
0024             pstr = plist;
0025             return
0026         elseif strcmp(in, 'Version')
0027             pstr = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 pstr = ['pole(' num2str(p.f)];
0034 if p.q > 0.5
0035     pstr = [pstr ',' num2str(p.q)];
0036 end
0037 pstr = [pstr ')'];
0038
0039 pstr = strrep(pstr, '_', '\_');
0040
0041 % END

```


[Home](#) > [classes](#) > [@pole](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a pole from a pole xml file.

SYNOPSIS

```
function ps = xmlparse(p, xmlfile)
```

DESCRIPTION

XMLPARSE creates a pole from a pole xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a pole object into a string.

This function is called by:

- [pole](#) POLE construct a pole object.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a pole from a pole xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```

```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'pole')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```


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.
 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.
 xml	XML convert a zero object to an DOM XML object.
 xmladd	XMLADD add a zero object to the given node of the input XML DOM
 xmlparse	XMLPARSE creates a zero from a zero xml file.
 zero	ZERO construct a pole object.

Subsequent directories:

- CVS

- [private](#)

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@zero](#) > char.m

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 1.3 2008/01/02 17:58:51 ingo 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.
- [eq](#) EQ overloads the == operator for zero objects.
- [get](#) GET get zero properties.
- [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.
- [set](#) SET set a zero property.

- **subsasgn** SUBSASGN define index assignment for zero properties.
- **subsref** SUBSREF Define field name indexing for zero objects.
- **xmlparse** XMLPARSE creates a zero from a zero xml file.
- **zero** ZERO construct a pole object.

SOURCE CODE

```

0001 function pstr = char(p, varargin)
0002 % CHAR convert a zero object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a zero object into a string.
0007 %
0008 % CALL:          string = char(zero);
0009 %
0010 % VERSION:       $Id: char.m,v 1.3 2008/01/02 17:58:51 ingo Exp $
0011 %
0012 % HISTORY:       02-04-2007 Hewitson
0013 %                  Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: char.m,v 1.3 2008/01/02 17:58:51 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(p, 'zero') && ischar(varargin{1})
0022         in = char(varargin{1});
0023         if strcmp(in, 'Params')
0024             pstr = plist;
0025             return
0026         elseif strcmp(in, 'Version')
0027             pstr = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 pstr = ['zero(' num2str(p.f)];
0034 if p.q > 0.5
0035     pstr = [pstr ',' num2str(p.q)];
0036 end
0037 pstr = [pstr ')'];
0038
0039 pstr = strrep(pstr, '_', '\_');
0040
0041 % END

```


[Home](#) > [classes](#) > [@zero](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a zero from a zero xml file.

SYNOPSIS

function ps = xmlparse(p, xmlfile)

DESCRIPTION

XMLPARSE creates a zero from a zero xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a zero object into a string.

This function is called by:

- [zero](#) ZERO construct a pole object.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a zero from a zero xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```

```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'zero')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```

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.
 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.
 xml	XML convert a pzmodel object to an DOM XML object.
 xmladd	XMLADD add a pzmodel object to the given node of the input XML DOM
 xmlparse	XMLPARSE creates a pzmodel from a pzmodel xml file.

Subsequent directories:

- CVS
- [private](#)

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@pzmodel](#) > char.m

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 1.3 2008/01/07 18:13:10 ingo 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.
- [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.
- [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.
- **xmlparse** XMLPARSE creates a pzmodel from a pzmodel xml file.

SOURCE CODE

```

0001 function pstr = char(p, varargin)
0002 % CHAR convert a pzmodel object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a pzmodel object into a string.
0007 %
0008 % CALL:          string = char(pzmodel)
0009 %
0010 % VERSION:       $Id: char.m,v 1.3 2008/01/07 18:13:10 ingo Exp $
0011 %
0012 % HISTORY:       02-04-2007 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: char.m,v 1.3 2008/01/07 18:13:10 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(p, 'pzmodel') && ischar(varargin{1})
0022         in = char(varargin{1});
0023         if strcmp(in, 'Params')
0024             pstr = plist;
0025             return
0026         elseif strcmp(in, 'Version')
0027             pstr = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 pstr = ['pzmodel(' p.name ')'];
0034 pstr = strrep(pstr, '_', '\_');
0035
0036 % END

```

[Home](#) > [classes](#) > [@pzmodel](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a pzmodel from a pzmodel xml file.

SYNOPSIS

```
function ps = xmlparse(p, xmlfile)
```

DESCRIPTION

XMLPARSE creates a pzmodel from a pzmodel xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a pzmodel object into a string.

This function is called by:

- [pzmodel](#) PZMODEL constructor for pzmodel class.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a pzmodel from a pzmodel xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```

```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'pzmodel')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```

Contents

PURPOSE

%% class: ao %%%

SYNOPSIS

This is a script file.

DESCRIPTION

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: ao %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help ao/abs">ao/abs</a>                - overloads the Absolute
value 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.

[ao/det](matlab:help ao/det)

determinant function for Analysis objects.

[ao/diag](matlab:help ao/diag)

operator for Analysis Objects.

[ao/display](matlab:help ao/display)

display for analysis object.

[ao/eig](matlab:help ao/eig)

determinant function for Analysis objects.

[ao/eq](matlab:help ao/eq)

for analysis objects.

[ao/exp](matlab:help ao/exp)

operator for Analysis objects. Exponential.

[ao/export](matlab:help ao/export)

object to a text file.

[ao/fft](matlab:help ao/fft)

operator for Analysis objects.

[ao/filter](matlab:help ao/filter)

function for analysis objects.

[ao/filtfilt](matlab:help ao/filtfilt)

filtfilt function for analysis objects.

[ao/find](matlab:help ao/find)

satisfy the input query and return a new AO.

[ao/ge](matlab:help ao/ge)

analysis objects. Compare the y-axis values.

[ao/get](matlab:help ao/get)

[ao/gt](matlab:help ao/gt)

analysis objects. Compare the y-axis values.

[ao/hist](matlab:help ao/hist)

function (hist) of MATLAB for Analysis Objects.

[ao/imag](matlab:help ao/imag)

operator for Analysis objects.

[ao/index](matlab:help ao/index)

or matrix. This properly captures the history.

[ao/interp](matlab:help ao/interp)

values in the input AO(s) at new values specified by the input parameter list.

[ao/inv](matlab:help ao/inv)

function for Analysis Objects.

[ao/iplot](matlab:help ao/iplot)

analysis objects.

[ao/isfield](matlab:help ao/isfield)

field is one of the object properties.

[ao/join](matlab:help ao/join)

single AO.

[ao/le](matlab:help ao/le)

analysis objects. Compare the y-axis values.

[ao/len](matlab:help ao/len)

operator for Analysis objects. Length of the data samples.

[ao/ln](matlab:help ao/ln)

operator for Analysis objects. Natural logarithm.

[ao/log](matlab:help ao/log)

operator for Analysis objects. Natural logarithm.

[ao/log10](matlab:help ao/log10)

- overloads the
- overloads the diagonal
- implement terminal
- overloads the
- overloads the == operator
- overloads the exp
- export an analysis
- overloads the fft
- overrides the filter
- overrides the
- particular samples that
- overloads >= operator for
- get ao properties.
- overloads > operator for
- overloads the histogram
- overloads the imaginary
- index into an AO array
- interpolate the
- overloads the inverse
- plots an array of
- tests if the given
- multiple AOs into a
- overloads <= operator for
- overloads the length
- overloads the log
- overloads the log
- overloads the log10

operator for Analysis objects. Common (base 10) logarithm.

[ao/lt](matlab:help ao/lt)

- overloads < operator for

analysis objects. Compare the y-axis values.

[ao/md5](matlab:help ao/md5)

- computes an MD5 checksum

from an analysis objects.

[ao/mean](matlab:help ao/mean)

- overloads the mean

operator for Analysis objects. Compute the mean value.

[ao/median](matlab:help ao/median)

- overloads the median

operator for Analysis objects. Compute the median value.

[ao/minus](matlab:help ao/minus)

- overloads - operator

for analysis objects.

[ao/mpower](matlab:help ao/mpower)

- implements power

operator for analysis objects.

[ao/mrdivide](matlab:help ao/mrdivide)

- overloads /

operator for analysis objects.

[ao/mtimes](matlab:help ao/mtimes)

- overloads * operator

for analysis objects.

[ao/mux](matlab:help ao/mux)

- concatonate AOs into a

vector.

[ao/ne](matlab:help ao/ne)

- overloads the ~= operator

for analysis objects.

[ao/norm](matlab:help ao/norm)

- overloads the norm

operator for Analysis Objects.

[ao/phase](matlab:help ao/phase)

- overloads the

ltpda_phase operator for Analysis objects.

[ao/plot](matlab:help ao/plot)

- a simple plot of

analysis objects.

[ao/plot_old](matlab:help ao/plot_old)

- plots an array of

analysis objects.

[ao/plus](matlab:help ao/plus)

- implements addition

operator for two analysis objects.

[ao/polyfit](matlab:help ao/polyfit)

- overloads polyfit()

function of MATLAB for Analysis Objects.

[ao/power](matlab:help ao/power)

- implements power

operator for analysis objects.

[ao/rdivide](matlab:help ao/rdivide)

- overloads ./

operator for analysis objects.

[ao/real](matlab:help ao/real)

- overloads the real

operator for Analysis objects.

[ao/resample](matlab:help ao/resample)

- overloads resample

function for AOs.

[ao/save](matlab:help ao/save)

- overloads save operator

for analysis objects.

[ao/select](matlab:help ao/select)

- select particular

samples from the input AOs and return new AOs with only those samples.

[ao/set](matlab:help ao/set)

- set an analysis object

property.

[ao/setnh](matlab:help ao/setnh)

- set an analysis object

property without recording the history.

[ao/simple_plot](matlab:help ao/simple_plot)

- a simple plot of

analysis objects.

[ao/sin](matlab:help ao/sin)

- overloads the sin

operator for Analysis objects. Sine of argument in radians.

ao/split	- split an analysis object into the specified segments.
ao/sqrt	- overloads the sqrt operator for Analysis objects. Square root.
ao/stairs	- overloads the stairs function for Analysis Objects.
ao/std	- overloads the std operator for Analysis objects. Compute the standard deviation.
ao/string	- writes a command string that can be used to recreate the input Analysis object(s).
ao/submit	- A0(s) to the MySQL server.
ao/submit_fastinsert	- A0(s) to the MySQL server.
ao/subsasgn	- define index assignment for analysis object properties.
ao/subsref	- Define field name indexing for analysis objects.
ao/sum	- overloads the sum operator for Analysis objects. Compute the sum value.
ao/svd	- overloads the determinant function for Analysis objects.
ao/tag	- tag each input A0 with a proper ID number.
ao/tan	- overloads the tan operator for Analysis objects. Tangent of argument in radians.
ao/testA0	- checks that an analysis object has the correct class fields.
ao/times	- overloads .* operator for analysis objects.
ao/transpose	- overloads the .' operator for Analysis Objects.
ao/var	- overloads the var operator for Analysis objects. Compute the variance.
ao/xml	- convert an analysis object to an DOM XML object.
ao/xmladd	- add an analysis object to the given node of the input XML DOM
ao/xmlparse	- creates a ao from a ao xml file.

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

[Home](#) > [classes](#) > [@ao](#) > ao2m.m

ao2m

PURPOSE

AO2M converts an analysis object to an '.m' file based on the history contained in the analysis object.

SYNOPSIS

```
function ao2m(a, filename)
```

DESCRIPTION

A02M converts an analysis object to an '.m' file based on the history contained in the analysis object.

%%%

DESCRIPTION: A02M converts an analysis object to an '.m' file based on the history contained in the analysis object.

CALL: ao2m(a, 'filename');

VERSION: \$Id: ao2m.m,v 1.11 2007/11/14 16:30:18 ingo Exp \$

HISTORY: 08-02-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR overloads char() function for analysis objects.

This function is called by:

SOURCE CODE

```
0001 function ao2m(a, filename)
0002 % A02M converts an analysis object to an '.m' file based on the history
0003 % contained in the analysis object.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

0005 %
0006 % DESCRIPTION: AO2M converts an analysis object to an '.m' file based on the
0007 %                  history contained in the analysis object.
0008 %
0009 % CALL:           ao2m(a, 'filename');
0010 %
0011 % VERSION:        $Id: ao2m.m,v 1.11 2007/11/14 16:30:18 ingo Exp $
0012 %
0013 % HISTORY: 08-02-07 M Hewitson
0014 %              Creation
0015 %
0016 %%%%%%%%%%%
0017
0018 ALGONAME = mfilename;
0019 VERSION  = '$Id: ao2m.m,v 1.11 2007/11/14 16:30:18 ingo Exp $';
0020
0021 if length(a) > 1
0022     error('### I can only deal with one AO at a time.');
```

```

0023 end
0024
0025 % clear this m-file from memory
0026 clear(filename);
0027
0028 % convert the history to a set of commands
0029 cmds = hist2m(a.hist);
0030
0031 % Now write output file
0032 [path,name,ext,vers] = fileparts(filename);
0033 if isempty(ext)
0034     filename = [filename '.m'];
0035 end
0036 if ~strcmp(ext, '.m')
0037     error('### I can only write .m files.');
```

```

0038 end
0039
0040 d = a.data;
0041 fd = fopen(filename, 'w+');
0042 fprintf(fd, 'function a_out = %s\n\n', name);
0043 fprintf(fd, '%% %s \n', upper(filename));
0044 fprintf(fd, '%% \n');
0045 fprintf(fd, '%% \n');
0046 fprintf(fd, '%% written by %s / %s\n', ALGONAME, VERSION);
0047 fprintf(fd, '%% \n');
0048 fprintf(fd, '%% based on analysis object:\n');
0049 fprintf(fd, '%%           name: %s / %s\n', a.name, d.name);
0050 fprintf(fd, '%%           provenance: %s\n', char(a.provenance));
0051 fprintf(fd, '%%           description: %s\n', char(a.description));
0052 fprintf(fd, '%% original m-file: %s\n', a.mfilename);
0053 fprintf(fd, '%% \n');
0054 fprintf(fd, '%% \n');
0055 fprintf(fd, ' \n');
0056 fprintf(fd, ' \n');
```

```
0057 for c=length(cmds):-1:1
0058     if ~isempty(cmds{c})
0059         fprintf(fd, '%s\n', cmds{c});
0060     end
0061 end
0062 fprintf(fd, ' \n');
0063 fprintf(fd, ' \n');
0064 fprintf(fd, '%% END\n');
0065
0066 fclose(fd);
0067
0068 cd ('.');
0069
0070
```

[Home](#) > [classes](#) > [@ao](#) > attachm.m

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 1.4 2007/11/22 15:11:16 ingo 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

```
0001 function a = attachm(a, mfile)
0002 % ATTACHM attach the m file being executes to the analysis object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ATTACHM attach the m file being executes to the analysis object.
0007 %
```

```

0008 % CALL:          ao = attachm(ao, 'foo.m');
0009 %
0010 % VERSION:        $Id: attachm.m,v 1.4 2007/11/22 15:11:16 ingo Exp $
0011 %
0012 % HISTORY:        31-03-07 M Hewitson
0013 %                  Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: attachm.m,v 1.4 2007/11/22 15:11:16 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(a, 'ao') && ischar(mfile)
0022         in = char(mfile);
0023         if strcmp(in, 'Params')
0024             a = plist;
0025             return
0026         elseif strcmp(in, 'Version')
0027             a = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033
0034 [pathstr,name,ext,vers] = fileparts(mfile);
0035 if isempty(ext)
0036     mfile = [mfile '.m'];
0037 end
0038 a.mfile = textread(mfile,'%s','delimiter','\n','whitespace','');
0039 a.mfilename = mfile;
0040
0041

```

[Home](#) > [classes](#) > [@ao](#) > attachmdl.m

attachmdl

PURPOSE

ATTACHMDL attach an mdl file to the analysis object.

SYNOPSIS

function a = attachmdl(a)

DESCRIPTION

ATTACHMDL attach an mdl file to the analysis object.

%%%

DESCRIPTION: ATTACHMDL attach an mdl file to the analysis object.

CALL: a = sltpda_attach_mdl(a)

VERSION: \$Id: attachmdl.m,v 1.3 2007/06/20 15:19:54 ingo Exp \$

HISTORY: 31-03-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function a = attachmdl(a)
0002 % ATTACHMDL attach an mdl file to the analysis object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ATTACHMDL attach an mdl file to the analysis object.
0007 %
0008 % CALL:           a = sltpda_attach_mdl(a)
0009 %
```

```

0010 % VERSION:      $Id: attachmdl.m,v 1.3 2007/06/20 15:19:54 ingo Exp $
0011 %
0012 % HISTORY: 31-03-07 M Hewitson
0013 %           Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 mdlfile = gcs;
0018 % find(pl, 'mdlfile');
0019
0020 [pathstr,name,ext,vers] = fileparts(mdlfile);
0021 if isempty(ext)
0022     mdlfile = [mdlfile '.mdl'];
0023 end
0024
0025
0026 mdlfilein = textread(mdlfile,'%s','delimiter','\n','whitespace','');
0027 mdlout = '';
0028 for j=1:length(mdlfilein)
0029     mdlout = [mdlout '\n' mdlfilein{j}];
0030 end
0031 a.mdlfile = mdlout;
0032 a.mdlfilename = mdlfile;
0033
0034
0035 % END

```


[Home](#) > [classes](#) > [@ao](#) > cat.m

cat

PURPOSE

CAT concatonate AOs into a vector.

SYNOPSIS

function b = cat(varargin)

DESCRIPTION

CAT concatonate AOs into a vector.

%%%

DESCRIPTION: CAT concatonate AOs into a vector.

CALL: a = cat(a1, a2)

VERSION: \$Id: cat.m,v 1.5 2007/10/23 16:30:44 ingo Exp \$

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

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

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

```

0001 function b = cat(varargin)
0002 % CAT concatonate AOs into a vector.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CAT concatonate AOs into a vector.
0007 %
0008 % CALL:          a = cat(a1, a2)
0009 %
0010 % VERSION:       $Id: cat.m,v 1.5 2007/10/23 16:30:44 ingo Exp $
0011 %
0012 % The following call returns a parameter list object that contains the
0013 % default parameter values:
0014 %
0015 % >> pl = cat(ao, 'Params')
0016 %
0017 % HISTORY: 27-03-07 M Hewitson
0018 %          Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: cat.m,v 1.5 2007/10/23 16:30:44 ingo Exp $';
0023
0024 %% Check if this is a call for parameters
0025 if nargin == 2
0026     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0027         in = char(varargin{2});
0028         if strcmp(in, 'Params')
0029             b = getDefaultPL();
0030             return
0031         elseif strcmp(in, 'Version')
0032             b = VERSION;
0033             return
0034         end
0035     end
0036 end
0037
0038 %% capture input variable names
0039 invars = {};
0040 for j=1:nargin
0041     iname = inputname(j);
0042     if isempty(iname) & isnumeric(varargin{j})
0043         iname = num2str(varargin{j});
0044     elseif isempty(iname) & ischar(varargin{j})
0045         iname = varargin{j};
0046     end
0047     invars = [invars cellstr(iname)];
0048 end
0049

```

```
0050 b = [];  
0051 for j=1:nargin  
0052     if isa(varargin{j}, 'ao')  
0053         b = [b varargin{j}];  
0054     end  
0055 end  
0056  
0057 %% Get default params  
0058 function pl_default = getDefaultPL()  
0059  
0060     pl_default = plist();  
0061  
0062  
0063 % END
```

[Home](#) > [classes](#) > [@ao](#) > demux.m

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.

%%%

DESCRIPTION: DEMUX splits the input vector of AOs into a number of output AOs.

CALL: [a1,a2,...] = demux(as)

VERSION: \$Id: demux.m,v 1.5 2007/10/23 16:30:44 ingo Exp \$

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

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

HISTORY: 28-03-07 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

```

0001 function varargout = demux(as, params)
0002 % DEMUX splits the input vector of AOs into a number of output AOs.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DEMUX splits the input vector of AOs into a number of output AOs.
0007 %
0008 % CALL:          [a1,a2,...] = demux(as)
0009 %
0010 % VERSION:       $Id: demux.m,v 1.5 2007/10/23 16:30:44 ingo Exp $
0011 %
0012 % The following call returns a parameter list object that contains the
0013 % default parameter values:
0014 %
0015 % >> pl = demux(ao, 'Params')
0016 %
0017 % HISTORY: 28-03-07 M Hewitson
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: demux.m,v 1.5 2007/10/23 16:30:44 ingo Exp $';
0023
0024 %% Check if this is a call for parameters
0025 if nargin == 2
0026     if isa(as, 'ao') && ischar(params)
0027         in = char(params);
0028         if strcmp(in, 'Params')
0029             varargout{1} = getDefaultPL();
0030             return
0031         elseif strcmp(in, 'Version')
0032             varargout{1} = VERSION;
0033             return
0034         end
0035     end
0036 end
0037
0038 %% get number of input AOs
0039 n = numel(as);
0040
0041 if nargin > n
0042     error('### too many output arguments');
0043 elseif nargin > n
0044     error('### not enough output arguments');
0045 end
0046
0047 for j=1:nargout
0048     % map to output AO
0049     varargout{j} = as(j);
0050 end

```

```
0051
0052 %% Get default params
0053 function pL_default = getDefaultPL()
0054
0055     pL_default = plist();
0056
0057 % END
```

Generated on Tue 22-Jan-2008 10:39:13 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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(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 == 1,
 otherwise, result == 0.

VERSION: \$Id: eq.m,v 1.10 2007/11/23 12:11:08 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:

- **eq** EQ overloads the == operator for analysis objects.
- **ne** NE overloads the ~= operator for analysis objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002
0003 % EQ overloads the == operator for analysis objects.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: EQ overloads the == operator for analysis objects.
0008 %
0009 %           All fields are checked.
0010 %
0011 % CALL:      result = eq(c1,c2)
0012 %           result = eq(c1,c2,  exc_list)
0013 %           result = eq(c1,c2, 'property1', 'property2')
0014 %           result = eq(c1,c2, 'class/property', 'class/property')
0015 %
0016 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0017 %           result = eq(c1,c2, 'ao/name')
0018 %
0019 % INPUTS:    c1,c2      - input analysis objects
0020 %           exc_list - exception list
0021 %                   List of properties which are not checked.
0022 %
0023 % OUTPUTS:   If the two analysis objects are considered equal, result == 1,
0024 %           otherwise, result == 0.
0025 %
0026 % VERSION:   $Id: eq.m,v 1.10 2007/11/23 12:11:08 ingo Exp $
0027 %
0028 % HISTORY:   29-08-2007 M Hewitson
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032
0033 VERSION = '$Id: eq.m,v 1.10 2007/11/23 12:11:08 ingo Exp $';
0034
0035 % Check if this is a call for parameters or for the cvs-version number
0036 if nargin == 2
0037     if isa(c1, 'ao') && ischar(c2)
0038         in = char(c2);
0039         if strcmp(in, 'Params')
0040             result = plist();

```



```

0041     return
0042     elseif strcmp(in, 'Version')
0043         result = VERSION;
0044         return
0045     end
0046 end
0047 end
0048
0049 %% Check length of c1 and c2
0050 if length(c1) ~= length(c2)
0051     disp(sprintf('\nNOT EQUAL: The size of the ao''s'));
0052     result = 0;
0053     return
0054 end
0055
0056 %% Check class
0057 if ~strcmp(class(c1), class(c2))
0058     disp(sprintf('\nNOT EQUAL: The class of the ao''s'));
0059     result = 0;
0060     return
0061 end
0062
0063
0064 %% for each element in c1 and c2
0065 for jj = 1:numel(c1)
0066
0067     result = 1;
0068
0069     fields = fieldnames(c1(jj));
0070
0071     for ii = 1:length(fields)
0072         field = fields{ii};
0073
0074         ck_field = {field, ['ao/' field]};
0075
0076         % Is the field equal to history then add the name 'history'
0077         % to the exception list.
0078         if strcmp(field, 'hist')
0079             ck_field{end+1} = 'history';
0080             ck_field{end+1} = 'ao/history';
0081         end
0082
0083         %% Check fields
0084         if ~(any(ismember(ck_field, varargin)))
0085
0086             if isobject(c1(jj).(field))
0087                 if ~eq(c1(jj).(field), c2(jj).(field), varargin{:})
0088                     result = 0;
0089                     disp(sprintf('\nNOT EQUAL: %s.%s', class(c1(jj)), field));
0090                     return
0091                 end
0092             else

```

```
0093         if ~isequalwithhequalnans(c1(jj).(field), c2(jj).(field))
0094             result = 0;
0095             disp(sprintf('\nNOT EQUAL: %s.%s', class(c1(jj)), field));
0096             return
0097         end
0098     end
0099
0100 end
0101 end
0102
0103 end
0104
```

[Home](#) > [classes](#) > [@ao](#) > export.m

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 1.6 2007/10/24 17:35:28 ingo 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

```
0001 function varargout = export(a, fname)
0002 % EXPORT export an analysis object to a text file.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

0005 %
0006 % DESCRIPTION: EXPORT export an analysis object to a text file.
0007 %
0008 % CALL:          export(a, 'blah.txt');
0009 %
0010 % VERSION:       $Id: export.m,v 1.6 2007/10/24 17:35:28 ingo Exp $
0011 %
0012 % HISTORY: 30-03-07 M Hewitson
0013 %              Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: export.m,v 1.6 2007/10/24 17:35:28 ingo Exp $';
0018
0019 %% Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(a, 'ao') && ischar(fname)
0022         in = char(fname);
0023         if strcmp(in, 'Params')
0024             varargout{1} = plist();
0025             return
0026         elseif strcmp(in, 'Version')
0027             varargout{1} = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 if isa(a, 'ao')
0034
0035     d = a.data;
0036
0037     [x,y] = get_xy_values(a.data);
0038
0039     if isreal(y)
0040         out = [x y];
0041     else
0042         out = [x real(y) imag(y)];
0043     end
0044
0045     save(fname, 'out', '-ASCII', '-DOUBLE', '-TABS');
0046
0047 else
0048     error('### incorrect usage: first argument should be an A0.');
```

```

0051 % END
0052
```

[Home](#) > [classes](#) > [@ao](#) > [ge.m](#)

ge

PURPOSE

GE overloads `>=` operator for analysis objects. Compare the y-axis values.

SYNOPSIS

function a = ge(b,c)

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 1.5 2007/11/23 12:11:08 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

```
0001 function a = ge(b,c)
0002 % GE overloads >= operator for analysis objects. Compare the y-axis values.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GE overloads >= operator for analysis objects.
0007 %             Compare the y-axis values.
0008 %
0009 % CALL: a = b>=c;
0010 %
0011 % POSSIBLE VALUES: b: Analysis object
0012 %                   c: Analysis object or a scalar.
0013 %
0014 % VERSION: $Id: ge.m,v 1.5 2007/11/23 12:11:08 ingo Exp $
0015 %
0016 % HISTORY: 06-02-2007 M Hewitson
0017 %             Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: ge.m,v 1.5 2007/11/23 12:11:08 ingo Exp $';
0022
0023 % Check if this is a call for parameters or for the cvs-version number
0024 if nargin == 2
0025     if isa(b, 'ao') && ischar(c)
0026         in = char(c);
0027         if strcmp(in, 'Params')
0028             a = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             a = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 %% Create parameter list to get the y-axis values The value is not necessary.
0038 % It is only necessary that the value is set.
0039 pl = plist(param('ydata', 'x_xx_y'));
0040
0041 %% Get the data from analysis object
0042 [dummy, x] = get_xy_values(b.data, pl);
0043
0044 %% Get data from second input
0045 if isa(c, 'ao')
0046     [dummy, y] = get_xy_values(c.data, pl);
0047 else
0048     y = c*ones(size(x));
0049 end
0050
0051 %% Create Output
0052 a = x>=y;
0053
0054 % END

```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > gt.m

gt

PURPOSE

GT overloads > operator for analysis objects. Compare the y-axis values.

SYNOPSIS

function a = gt(b,c)

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 1.4 2007/11/23 12:11:08 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

```
0001 function a = gt(b,c)
0002 % GT overloads > operator for analysis objects. Compare the y-axis values.
0003 %
```



```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GT overloads > operator for analysis objects.
0007 %             Compare the y-axis values.
0008 %
0009 % CALL: a = b>=;
0010 %
0011 % POSSIBLE VALUES: b: Analysis object
0012 %                   c: Analysis object or a scalar.
0013 %
0014 % VERSION: $Id: gt.m,v 1.4 2007/11/23 12:11:08 ingo Exp $
0015 %
0016 % HISTORY: 06-02-2007 M Hewitson
0017 %             Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: gt.m,v 1.4 2007/11/23 12:11:08 ingo Exp $';
0022
0023 % Check if this is a call for parameters or for the cvs-version number
0024 if nargin == 2
0025     if isa(b, 'ao') && ischar(c)
0026         in = char(c);
0027         if strcmp(in, 'Params')
0028             a = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             a = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 %% Create parameter list to get the y-axis values
0038 pl = plist(param('ydata', 'x_xx_y'));
0039
0040 %% Get the data from analysis object
0041 [dummy, x] = get_xy_values(b.data, pl);
0042
0043 %% Get data from second input
0044 if isa(c, 'ao')
0045     [dummy, y] = get_xy_values(c.data, pl);
0046 else
0047     y = c*ones(size(x));
0048 end
0049
0050 %% Create Output
0051 a = x>y;
0052
0053 % END

```


[Home](#) > [classes](#) > [@ao](#) > index.m

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 1.6 2007/11/02 12:26:24 ingo Exp \$

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

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

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

```

0001 function b = index(varargin)
0002 % INDEX index into an AO array or matrix. This properly captures the history.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: INDEX into an AO array or matrix. This properly captures the
0007 %               history. This function is also called when standard () indexing
0008 %               is used for AO arrays.
0009 %
0010 % CALL:         b = index(a, i)
0011 %               b = index(a, i, j)
0012 %
0013 % VERSION:      $Id: index.m,v 1.6 2007/11/02 12:26:24 ingo Exp $
0014 %
0015 % The following call returns a parameter list object that contains the
0016 % default parameter values:
0017 %
0018 % >> pl = index(ao, 'Params')
0019 %
0020 % HISTORY: 02-05-07 M Hewitson
0021 %           Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 ALGONAME = mfilename;
0026 VERSION  = '$Id: index.m,v 1.6 2007/11/02 12:26:24 ingo Exp $';
0027
0028 %% Check if this is a call for parameters
0029 if nargin == 2
0030     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0031         in = char(varargin{2});
0032         if strcmp(in, 'Params')
0033             b = getDefaultPL();
0034             return
0035         elseif strcmp(in, 'Version')
0036             b = VERSION;
0037             return
0038         end
0039     end
0040 end
0041
0042 %% capture input variable names

```

```

0043 invars = {};
0044 for j=1:nargin
0045     invars = [invars cellstr(inputname(j))];
0046 end
0047
0048 as = varargin{1};
0049 if ~isa(as, 'ao')
0050     error('### an array or matrix of AO objects is expected as first input
argument.');
```

```

0051 end
0052
0053 % Get indices
0054 pl = [];
0055 if nargin == 2
0056     if isa(varargin{2}, 'plist')
0057         pl = varargin{2};
0058         i = find(pl, 'i');
0059         j = find(pl, 'j');
0060         % Index the AO array
0061         if isempty(j)
0062             % Index the AO array
0063             asi = as(i);
0064         else
0065             % Index the AO array
0066             asi = as(i,j);
0067         end
0068     else
0069         i = varargin{2};
0070         j = 0;
0071         % Index the AO array
0072         asi = as(i);
0073     end
0074 elseif nargin == 3
0075     i = varargin{2};
0076     j = varargin{3};
0077     if j == 0
0078         % Index the AO array
0079         asi = as(i);
0080     else
0081         % Index the AO array
0082         asi = as(i,j);
0083     end
0084 else
0085     error('### incorrect inputs.');
```

```

0086 end
0087
0088
0089 si = size(asi);
0090 na = si(1)*si(2);
0091 if na ~= 1
0092     error('### I only support a single index currently.');
```

```

0093 end

```

```
0094
0095 b = [];
0096
0097 % Now build output AO
0098 if isempty(pl)
0099     pl = plist([param('i', i) param('j', j)]);
0100 end
0101 h = history(ALGONAME, VERSION, pl, get(asi, 'hist'));
0102 h = set(h, 'invars', invars);
0103
0104 % make output analysis object
0105 aout = setnh(asi, 'hist', h);
0106
0107 b = [b aout];
0108
0109 %% Get default params
0110 function pl_default = getDefaultPL()
0111
0112     pl_default = plist();
0113     pl_default = append(pl_default, 'i', []);
0114     pl_default = append(pl_default, 'j', []);
0115
0116
0117 % END
```

[Home](#) > [classes](#) > [@ao](#) > iplot.m

iplot

PURPOSE

IPLLOT provides an intelligent plotting tool for LTPDA.

SYNOPSIS

function varargout = iplot(varargin)

DESCRIPTION

IPLLOT provides an intelligent plotting tool for LTPDA.

%%%

DESCRIPTION: IPLLOT provides an intelligent plotting tool for LTPDA.

CALL:

```

        hfig = iplot (a,pl)
    [hfig, hax] = iplot (a,pl)
[hfig, hax, hli] = iplot (a,pl)

```

INPUTS:

- pl - a parameter list
- a - input analysis object

OUTPUTS:

- hfig - handles to figures
- hax - handles to axes
- hli - handles to lines

Plot parameters:

'type' - one of the data class types
'tsdata' [default], 'fsdata', 'cdata', 'xydata', 'xyzdata'.

'Arrangement' - select the plot layout:
 'single' - plots all AOs on individual figures
 'stacked' - plots all AOs on the same axes [default]
 'subplots' - plots all AOs on subplots

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.

'LineColors' - a cell array of color definitions for each line.

'LineStyles' - a cell array of line styles.

'Markers' - a cell array of markers.

'LineWidths' - an array of line widths. If the length of the 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.

'XScales' - Specify the scales to be used on the x-axes.

'YScales' - Specify the scales to be used on the y-axes. If an object contains complex data, you should specify two y-labels for that object.

VERSION: \$Id: iplot.m,v 1.17 2008/01/21 11:40:25 hewitson Exp \$

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

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

HISTORY: 22-12-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- **abs** ABS overloads the Absolute value operator for Analysis objects.
- **char** CHAR overloads char() function for analysis objects.
- **find** FIND particular samples that satisfy the input query and return a new AO.
- **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

```

0001 function varargout = iplot(varargin)
0002 % IPLLOT provides an intelligent plotting tool for LTPDA.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: IPLLOT provides an intelligent plotting tool for LTPDA.
0007 %
0008 % CALL:
0009 %             hfig = iplot (a,pl)
0010 %             [hfig, hax] = iplot (a,pl)
0011 %             [hfig, hax, hli] = iplot (a,pl)
0012 %
0013 % INPUTS:
0014 %             pl - a parameter list
0015 %             a - input analysis object
0016 %
0017 % OUTPUTS:
0018 %             hfig - handles to figures
0019 %             hax - handles to axes
0020 %             hli - handles to lines
0021 %
0022 % Plot parameters:
0023 %
0024 %             'type' - one of the data class types
0025 %                     'tsdata' [default], 'fsdata', 'cdata', 'xydata',
0026 %                     'xyzdata'.
0027 %
0028 %             'Arrangement' - select the plot layout:
0029 %                     'single' - plots all AOs on individual figures

```

```

0026 %             'stacked' - plots all AOs on the same axes
[default]
0027 %             'subplots' - plots all AOs on subplots
0028 %
0029 % Line parameters:
0030 %
0031 %     The following properties take cell array values. If the length of
0032 %     the cell array is shorter than the number of lines to plot, the
0033 %     remaining lines will be plotted with the default options.
0034 %
0035 %         'LineColors' - a cell array of color definitions for each line.
0036 %
0037 %         'LineStyles' - a cell array of line styles.
0038 %
0039 %         'Markers'     - a cell array of markers.
0040 %
0041 %         'LineWidths' - an array of line widths. If the length of the
0042 %         array is shorter than the number of lines to
0043 %         plot, the remaining lines will be plotted with
0044 %         the default line width.
0045 %
0046 % Axes parameters:
0047 %
0048 %         'Legends' - specify a cell array of strings to be used for
0049 %         the plot legends. If a cell contains an empty
0050 %         string, the default legend string is built.
0051 %         If a single string 'off' is given instead of a
0052 %         cell array, then the legends are all switched
0053 %         off.
0054 %
0055 %         'XLabels' - Specify the labels to be used on the x-axis. The
0056 %         units are added from the data object 'xunits'
0057 %         property.
0058 %
0059 %         'YLabels' - Specify the labels to be used on the y-axis. The
0060 %         units are added from the data object 'yunits'
0061 %         property. If the object contains complex data,
0062 %         you should specify two y-labels for that object.
0063 %
0064 %         'XScales' - Specify the scales to be used on the x-axes.
0065 %
0066 %         'YScales' - Specify the scales to the used on the y-axes. If
0067 %         an object contains complex data, you should
0068 %         specify two y-labels for that object.
0069 %
0070 %
0071 %
0072 % VERSION:      $Id: iplot.m,v 1.17 2008/01/21 11:40:25 hewitson Exp $
0073 %
0074 % The following call returns a parameter list object that contains the
0075 % default parameter values:
0076 %
0077 % >> pl = iplot(ao, 'Params')

```

```

0078 %
0079 % HISTORY: 22-12-07 M Hewitson
0080 %           Creation
0081 %
0082 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0083
0084
0085 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0086 %
0087 % TODO:
0088 %     1) Add XRange, YRange, ZRange
0089 %     2) What about complex cdata objects?
0090 %     3) What about cdata matrices ? Need to process all added lines.
0091 %     4) What about time-series with different t0's.
0092 %     5) How do we deal with axis units? Symbolic math toolbox?
0093 %     6) We could have 'All' properties. E.g. plist('AllMarkers', 'o')
0094 %     7) Should we have a plot_plist field in A0? Then this can be combined
0095 %         with the input plist.
0096 %
0097
0098 %% Check if this is a call for parameters
0099
0100 VERSION = '$Id: iplot.m,v 1.17 2008/01/21 11:40:25 hewitson Exp $';
0101 bs      = [];
0102
0103 if nargin == 2
0104     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0105         in = char(varargin{2});
0106         if strcmp(in, 'Params')
0107             varargout{1} = getDefaultPlist();
0108             return
0109         elseif strcmp(in, 'Version')
0110             varargout{1} = VERSION;
0111             return
0112         end
0113     end
0114 end
0115
0116 %% Collect input ao's, plist's and ao variable names
0117 in_names = {};
0118 for ii = 1:nargin
0119     in_names{end+1} = inputname(ii);
0120 end
0121
0122 [as, upl, invars] = collect_inputs(varargin, in_names);
0123
0124 %% Go through AOs and collect them into similar types
0125
0126 tsAOs = [];
0127 fsAOs = [];
0128 xyAOs = [];
0129 xyzAOs = [];
0130 cAOs = [];

```

```

0131
0132 for j=1:numel(as)
0133     switch class(as(j).data)
0134         case 'tsdata'
0135             tsA0s = [tsA0s as(j)];
0136         case 'fsdata'
0137             fsA0s = [fsA0s as(j)];
0138         case 'xydata'
0139             xyA0s = [xyA0s as(j)];
0140         case 'xyzdata'
0141             xyzA0s = [xyzA0s as(j)];
0142         case 'cdata'
0143             cA0s = [cA0s as(j)];
0144         otherwise
0145             warning('!!! Unknown data type %s', class(as(j).data));
0146     end
0147 end
0148
0149 %% Now plot all the objects on separate figures
0150
0151 hfig = [];
0152 hax  = [];
0153 hli  = [];
0154
0155 %----- TSDATA
0156
0157 % get default plist
0158 dpl = getDefaultPlist('tsdata');
0159 % combine the plists
0160 pl = combine(upl, dpl);
0161 % Call x-y plot
0162 [hf, ha, hl] = xy_plot(tsA0s, pl);
0163 hfig = [hfig hf];
0164 hax  = [hax ha];
0165 hli  = [hli hl];
0166
0167 %----- XYDATA
0168
0169 % get default plist
0170 dpl = getDefaultPlist('xydata');
0171 % combine the plists
0172 pl = combine(upl, dpl);
0173 % Call x-y plot
0174 [hf, ha, hl] = xy_plot(xyA0s, pl);
0175 hfig = [hfig hf];
0176 hax  = [hax ha];
0177 hli  = [hli hl];
0178
0179 %----- XYZDATA
0180
0181 % get default plist
0182 dpl = getDefaultPlist('xyzdata');

```

```

0183 % combine the plists
0184 pl = combine(upl, dpl);
0185 % Call x-y-z plot
0186 [hf, ha, hl] = xyz_plot(xyzA0s, pl);
0187 hfig = [hfig hf];
0188 hax   = [hax ha];
0189 hli   = [hli hl];
0190
0191 %----- CDATA
0192
0193 % get default plist
0194 dpl = getDefaultPlist('cddata');
0195 % combine the plists
0196 pl = combine(upl, dpl);
0197 % Call x-y plot
0198 [hf, ha, hl] = y_plot(cA0s, pl);
0199 hfig = [hfig hf];
0200 hax   = [hax ha];
0201 hli   = [hli hl];
0202
0203 %----- FSDATA
0204
0205 % get default plist
0206 dpl = getDefaultPlist('fsdata');
0207 % combine the plists
0208 pl = combine(upl, dpl);
0209 % Call fsdata plot
0210 [hf, ha, hl] = fs_plot(fsA0s, pl);
0211 hfig = [hfig hf];
0212 hax   = [hax ha];
0213 hli   = [hli hl];
0214
0215 %% Deal with outputs
0216 if nargin == 1
0217     varargout{1} = hfig;
0218 end
0219 if nargin == 2
0220     varargout{1} = hfig;
0221     varargout{2} = hax;
0222 end
0223 if nargin == 3
0224     varargout{1} = hfig;
0225     varargout{2} = hax;
0226     varargout{3} = hli;
0227 end
0228
0229 if nargin > 3
0230     error('### Incorrect number of outputs');
0231 end
0232
0233
0234 %-----

```

```

0235 % Plot fsdata objects
0236 %
0237 function varargout = fs_plot(varargin)
0238
0239 aos = varargin{1};
0240 pl = varargin{2};
0241
0242 % Extract parameters
0243 arrangement = find(pl, 'Arrangement');
0244 linecolors = find(pl, 'LineColors');
0245 colors = find(pl, 'Colors');
0246 linestyles = find(pl, 'LineStyles');
0247 markers = find(pl, 'Markers');
0248 linewidths = find(pl, 'LineWidths');
0249 legends = find(pl, 'Legends');
0250 ylabel = find(pl, 'YLabels');
0251 xlabel = find(pl, 'XLabels');
0252 yscale = find(pl, 'YScales');
0253 xscale = find(pl, 'XScale');
0254 yrange = find(pl, 'YRanges');
0255 xrange = find(pl, 'XRanges');
0256 xmath = find(pl, 'XMaths');
0257 ymath = find(pl, 'YMaths');
0258 complexPlotType = find(pl, 'complexPlotType');
0259
0260 % check whether we want legends or not
0261 if iscell(legends)
0262     legendsOn = 1;
0263 else
0264     if strcmpi(legends, 'off')
0265         legendsOn = 0;
0266     else
0267         legendsOn = 1;
0268         legends = [];
0269     end
0270 end
0271
0272 if ~iscell(linestyles), linestyles = {linestyles}; end
0273 if ~iscell(linecolors), linecolors = {linecolors}; end
0274 if ~iscell(markers), markers = {markers}; end
0275 if ~iscell(legends), legends = {legends}; end
0276 if ~iscell(ylabel), ylabel = {ylabel}; end
0277 if ~iscell(xlabel), xlabel = {xlabel}; end
0278 if ~iscell(xmath), xmath = {xmath}; end
0279 if ~iscell(ymath), ymath = {ymath}; end
0280 if ~iscell(xscale), xscale = {xscale}; end
0281 if ~iscell(yscale), yscale = {yscale}; end
0282 if ~iscell(xrange), xrange = {xrange}; end
0283 if ~iscell(yrange), yrange = {yrange}; end
0284
0285

```

```

0286 % collect figure handles
0287 tsfig = [];
0288 tsax  = [];
0289 tsli  = [];
0290
0291 % Legend holder
0292 legendStr = [];
0293
0294 if ~isempty(aos)
0295
0296     % Now loop over AOs
0297     Na = length(aos);
0298
0299     % - first to check if any are complex
0300     haveComplex = 0;
0301     for j=1:Na
0302         a = aos(j);
0303         y = a.data.y;
0304         ymath = '';
0305         if j<=length(ymaths)
0306             if ~isempty(ymaths{j})
0307                 eval(sprintf('y = %s;', ymaths{j}));
0308                 ymath = ymaths{j};
0309             end
0310         end
0311         xmath = '';
0312         if j<=length(xmaths)
0313             if ~isempty(xmaths{j})
0314                 eval(sprintf('x = %s;', xmaths{j}));
0315                 xmath = xmaths{j};
0316             end
0317         end
0318         if ~isreal(y)
0319             haveComplex = 1;
0320         end
0321     end
0322
0323     % Loop over the AOs now
0324     for j=1:Na
0325
0326         % set real and imag subplot handles to empty
0327         tsax_r = [];
0328         tsax_i = [];
0329
0330         % Get this AO
0331         a = aos(j);
0332
0333         %----- Apply math functions
0334         x = a.data.x;
0335         y = a.data.y;
0336
0337         ymath = '';
0338         if j<=length(ymaths)

```

```

0339         if ~isempty(ymaths{j})
0340             eval(sprintf('y = %s;', ymaths{j}));
0341             ymath = ymaths{j};
0342         end
0343     end
0344     xmath = '';
0345     if j<=length(xmaths)
0346         if ~isempty(xmaths{j})
0347             eval(sprintf('x = %s;', xmaths{j}));
0348             xmath = xmaths{j};
0349         end
0350     end
0351
0352     % what figures do we need?
0353     switch arrangement
0354     case 'single'
0355
0356         tsfig = [tsfig figure];
0357         col = colors{1};
0358         % check if this data set is real or complex
0359         if ~isreal(y)
0360             % complex means we use two subplots
0361             tsax_r = subplot(2,1,1);
0362             tsax_i = subplot(2,1,2);
0363             tsax    = [tsax tsax_r tsax_i];
0364         else
0365             % real means we use a single subplot
0366             tsax_r = subplot(1, 1, 1);
0367             tsax = [tsax tsax_r];
0368         end
0369
0370     case 'stacked'
0371
0372         if j==1, tsfig = figure; end
0373         % if at least one of the input fsdata AOs is complex, we need to
0374         % allow for subplots
0375         if haveComplex
0376             tsax_r = subplot(2,1,1);
0377             tsax_i = subplot(2,1,2);
0378             tsax    = [tsax_r tsax_i];
0379         else
0380             tsax_r = subplot(1, 1, 1);
0381             tsax = tsax_r;
0382         end
0383         col = colors{mod(j-1,length(colors))+1};
0384         hold(tsax_r, 'on');
0385         if ishandle(tsax_i)
0386             hold(tsax_i, 'on');
0387         end
0388     case 'subplots'
0389
0390         if j == 1, tsfig = figure; end
0391         c = 1+(j-1)*2;

```



```

0392     sx = Na;
0393     sy = 2;
0394     % Now we have one or two subplots per input object.
0395     if ~isreal(y)
0396         tsax_r = subplot(sx, sy,c);
0397         tsax_i = subplot(sx, sy,c+1);
0398         tsax    = [tsax tsax_r tsax_i];
0399     else
0400         tsax_r = subplot(sx, sy, c:c+1);
0401         tsax    = [tsax tsax_r];
0402     end
0403     col = colors{1};
0404
0405     otherwise
0406         error('### Unknown plot arrangement');
0407     end
0408
0409     %----- Plot the data
0410
0411     % plot real or complex data and setup default values for scale and
0412     % labels as we go.
0413     if isreal(y)
0414         tsli = [tsli plot(tsax_r, x, y)];
0415         ylabelr = 'amplitude';
0416         ylabeli = 'imag';
0417         yscaleR = 'log';
0418         yscaleI = 'lin';
0419         xscaleR = 'log';
0420         xscaleI = 'lin';
0421     else
0422         switch complexPlotType
0423             case 'realimag'
0424                 tsli = [tsli plot(tsax_r, x, real(y))];
0425                 tsli = [tsli plot(tsax_i, x, imag(y))];
0426                 ylabelr = 'real';
0427                 ylabeli = 'imag';
0428                 yscaleR = 'lin';
0429                 yscaleI = 'lin';
0430                 xscaleR = 'lin';
0431                 xscaleI = 'lin';
0432             case 'absdeg'
0433                 tsli = [tsli plot(tsax_r, x, abs(y))];
0434                 tsli = [tsli plot(tsax_i, x, ltpda_phase(y))];
0435                 ylabelr = 'amplitude';
0436                 ylabeli = 'Phase [deg]';
0437                 yscaleR = 'log';
0438                 yscaleI = 'lin';
0439                 xscaleR = 'log';
0440                 xscaleI = 'log';
0441             case 'absrad'
0442                 tsli = [tsli plot(tsax_r, x, abs(y))];
0443                 tsli = [tsli plot(tsax_i, x, angle(y))];

```

```

0444         ylabelr = 'amplitude';
0445         ylabeli = 'phase [rad]';
0446         yscaleR = 'log';
0447         yscaleI = 'lin';
0448         xscaleR = 'log';
0449         xscaleI = 'log';
0450     otherwise
0451         error('### Unknown plot type for complex data');
0452     end
0453 end
0454
0455 %----- Axis properties
0456
0457 % Set ylabel
0458 c = 1+(j-1)*2;
0459 if c<=length(ylabels)
0460     if ~isempty(ylabels{c})
0461         ylstrR = ylabels{c};
0462     else
0463         ylstrR = ylabelr;
0464     end
0465 else
0466     ylstrR = ylabelr;
0467 end
0468 if c<length(ylabels)
0469     if ~isempty(ylabels{c+1})
0470         ylstrI = ylabels{c+1};
0471     else
0472         ylstrI = ylabeli;
0473     end
0474 else
0475     ylstrI = ylabeli;
0476 end
0477 if ~isempty(ymath)
0478     ymath = strrep(ymath, 'y', sprintf('%s', a.data.yunits));
0479     ylstrR = [ylstrR ' [' ymath '] '];
0480 else
0481     ylstrR = [ylstrR ' [' a.data.yunits '] '];
0482 end
0483 ylabel(tsax_r, ylstrR);
0484 if ishandle(tsax_i)
0485     ylabel(tsax_i, ylstrI);
0486 end
0487
0488 % Set xlabel
0489 if j<=length(xlabels) && ~isempty(xlabels{j})
0490     xlstr = xlabels{j};
0491 else
0492     xlstr = find(pl, 'XLabel');
0493 end
0494 if ~isempty(xmath)
0495     xmath = strrep(xmath, 'x', sprintf('%s', a.data.xunits));
0496     xlstr = [xlstr ' [' xmath '] '];

```

```

0497     else
0498         xlstr = [xlstr ' [' a.data.xunits ']' ];
0499     end
0500     xlabel(tsax_r, xlstr);
0501     if ishandle(tsax_i)
0502         xlabel(tsax_i, xlstr);
0503     end
0504
0505     % Set grid on or off
0506     grid(tsax_r, 'on');
0507     if ishandle(tsax_i)
0508         grid(tsax_i, 'on');
0509     end
0510
0511     % Set Y scale
0512     c = 1+(j-1)*2;
0513     if c<=length(yscales)
0514         if ~isempty(yscales{c})
0515             yscaleR = yscales{c};
0516         end
0517     end
0518     if c<length(yscales)
0519         if ~isempty(yscales{c+1})
0520             yscaleI = yscales{c+1};
0521         end
0522     end
0523     set(tsax_r, 'YScale', yscaleR);
0524     if ishandle(tsax_i)
0525         set(tsax_i, 'YScale', yscaleI);
0526     end
0527
0528     % Set X scale
0529     c = 1+(j-1)*2;
0530     if c<=length(xscales)
0531         if ~isempty(xscales{c})
0532             xscaleR = xscales{c};
0533         end
0534     end
0535     if c<length(xscales)
0536         if ~isempty(xscales{c+1})
0537             xscaleI = xscales{c+1};
0538         end
0539     end
0540     set(tsax_r, 'XScale', xscaleR);
0541     if ishandle(tsax_i)
0542         set(tsax_i, 'XScale', xscaleI);
0543     end
0544
0545     % Set Y range
0546     c = 1+(j-1)*2;
0547     if c<=length(yranges)
0548         if ~isempty(yranges{c})

```

```

0549         set(tsax_r, 'YLim', yranges{c});
0550     end
0551 end
0552 if c<length(yranges)
0553     if ~isempty(yranges{c+1})
0554         if ishandle(tsax_i)
0555             set(tsax_i, 'YLim', yranges{c+1});
0556         end
0557     end
0558 end
0559
0560 % Set X range
0561 c = 1+(j-1)*2;
0562 if c<=length(xranges)
0563     if ~isempty(xranges{c})
0564         set(tsax_r, 'XLim', xrange{c});
0565     end
0566 end
0567 if c<length(xranges)
0568     if ~isempty(xranges{c+1})
0569         if ishandle(tsax_i)
0570             set(tsax_i, 'XLim', xrange{c+1});
0571         end
0572     end
0573 end
0574
0575 %----- line properties
0576
0577 % Set line color
0578 if isreal(y)
0579     set(tqli(end), 'Color', col);
0580 else
0581     set(tqli(end-1), 'Color', col);
0582     set(tqli(end), 'Color', col);
0583 end
0584
0585 % Override with user colors
0586 if j<=length(linecolors) && ~isempty(linecolors{j})
0587     if isreal(y)
0588         set(tqli(end), 'Color', linecolors{j});
0589     else
0590         set(tqli(end-1), 'Color', linecolors{j});
0591         set(tqli(end), 'Color', linecolors{j});
0592     end
0593 end
0594
0595 % Set line style
0596 if j<=length(linestyles) && ~isempty(linestyles{j})
0597     if isreal(y)
0598         set(tqli(end), 'LineStyle', linestyles{j});
0599     else

```

```

0600         set(tqli(end-1), 'LineStyle', linestyle{j});
0601         set(tqli(end), 'LineStyle', linestyle{j});
0602     end
0603 end
0604
0605 % Set line widths
0606 if j<=length(linewidths) && ~isempty(linewidths{j})
0607     if isreal(y)
0608         set(tqli(end), 'LineWidth', linewidths{j});
0609     else
0610         set(tqli(end-1), 'LineWidth', linewidths{j});
0611         set(tqli(end), 'LineWidth', linewidths{j});
0612     end
0613 end
0614
0615 % Set markers
0616 if j<=length(markers) && ~isempty(markers{j})
0617     if isreal(y)
0618         set(tqli(end), 'Marker', markers{j});
0619     else
0620         set(tqli(end-1), 'Marker', markers{j});
0621         set(tqli(end), 'Marker', markers{j});
0622     end
0623 end
0624
0625 % Set legend string
0626 lstr = '';
0627 if legendsOn
0628     if j<=length(legends) && ~isempty(legends{j})
0629         lstr = legends{j};
0630     else
0631         lstr = ltpda_label(a.name);
0632     end
0633 end
0634 legendStr = [legendStr cellstr(lstr)];
0635
0636 % Set the legend now if we can
0637 if legendsOn
0638     if strcmp(arrangement, 'single') || strcmp(arrangement, 'subplots')
0639         legend(tsax_r, legendStr{end});
0640     end
0641 end
0642
0643 end % End loop over AOs
0644
0645 % Process legends for stacked plots
0646 if legendsOn
0647     if strcmp(arrangement, 'stacked')
0648         h = legend(tsax_r, legendStr);
0649         set(h, 'FontSize', 10)
0650     end
0651

```

```

0652     end
0653
0654 end
0655
0656 % Set outputs
0657 if nargin > 0
0658     varargout{1} = tsfig;
0659 end
0660 if nargin > 1
0661     varargout{2} = tsax;
0662 end
0663 if nargin == 3
0664     varargout{3} = tsli;
0665 end
0666 if nargin > 3
0667     error('### Too many output arguments');
0668 end
0669
0670 %-----
0671 % Plot tsdata and xydata objects
0672 %
0673 function varargout = xy_plot(varargin)
0674
0675 aos = varargin{1};
0676 pl  = varargin{2};
0677
0678 % Extract parameters
0679 arrangement = find(pl, 'Arrangement');
0680 linecolors  = find(pl, 'LineColors');
0681 colors      = find(pl, 'Colors');
0682 linestyles  = find(pl, 'LineStyles');
0683 markers     = find(pl, 'Markers');
0684 linewidths  = find(pl, 'LineWidths');
0685 legends     = find(pl, 'Legends');
0686 ylabel      = find(pl, 'YLabels');
0687 xlabel      = find(pl, 'XLabels');
0688 xmaths      = find(pl, 'XMaths');
0689 ymaths      = find(pl, 'YMaths');
0690
0691 % check whether we want legends or not
0692 if iscell(legends)
0693     legendsOn = 1;
0694 else
0695     if strcmpi(legends, 'off')
0696         legendsOn = 0;
0697     else
0698         legendsOn = 1;
0699         legends = [];
0700     end
0701 end
0702

```

```

0703 if ~iscell(linestyles), linestyles = {linestyles}; end
0704 if ~iscell(linecolors), linecolors = {linecolors}; end
0705 if ~iscell(markers), markers = {markers}; end
0706 if ~iscell(legends), legends = {legends}; end
0707 if ~iscell(ylabels), ylabels = {ylabels}; end
0708 if ~iscell(xlabels), xlabels = {xlabels}; end
0709 if ~iscell(xmaths), xmaths = {xmaths}; end
0710 if ~iscell(ymaths), ymaths = {ymaths}; end
0711
0712
0713 % collect figure handles
0714 tsfig = [];
0715 tsax  = [];
0716 tsli  = [];
0717
0718 % Legend holder
0719 legendStr = [];
0720
0721 if ~isempty(aos)
0722
0723     % Now loop over AOs
0724     Na = length(aos);
0725     for j=1:Na
0726
0727         % Get this AO
0728         a = aos(j);
0729
0730         % what figures do we need?
0731         switch arrangement
0732             case 'single'
0733
0734                 tsfig = [tsfig figure];
0735                 tsax = subplot(1,1,1);
0736                 col = colors{1};
0737
0738             case 'stacked'
0739
0740                 if j==1, tsfig = figure; end
0741                 tsax = subplot(1,1,1);
0742                 col = colors{mod(j-1,length(colors))+1};
0743                 hold on;
0744
0745             case 'subplots'
0746
0747                 if j == 1, tsfig = figure; end
0748                 tsax = [tsax subplot(Na, 1, j)];
0749                 col = colors{1};
0750
0751             otherwise
0752                 error('### Unknown plot arrangement');
0753         end
0754
0755         %----- Apply math functions

```

```

0756     x = a.data.x;
0757     y = a.data.y;
0758
0759     ymath = '';
0760     if j<=length(ymaths)
0761         if ~isempty(ymaths{j})
0762             eval(sprintf('y = %s;', ymaths{j}));
0763             ymath = ymaths{j};
0764         end
0765     end
0766     xmath = '';
0767     if j<=length(xmaths)
0768         if ~isempty(xmaths{j})
0769             eval(sprintf('x = %s;', xmaths{j}));
0770             xmath = xmaths{j};
0771         end
0772     end
0773
0774     %----- Plot the data
0775
0776     tsli = [tsli plot(tsax(end), x, y)];
0777
0778     %----- Axis properties
0779
0780     % Set ylabel
0781     if j<=length(ylabels) && ~isempty(ylabels{j})
0782         ylstr = ylabels{j};
0783     else
0784         ylstr = find(pl, 'YLabel');
0785     end
0786     if ~isempty(ymath)
0787         ymath = strrep(ymath, 'y', sprintf('%s', a.data.yunits));
0788         ylstr = [ylstr ' [' ymath '] '];
0789     else
0790         ylstr = [ylstr ' [' a.data.yunits '] '];
0791     end
0792     ylabel(ylstr);
0793
0794     % Set xlabel
0795     if j<=length(xlabels) && ~isempty(xlabels{j})
0796         xlstr = xlabels{j};
0797     else
0798         xlstr = find(pl, 'XLabel');
0799     end
0800     if ~isempty(xmath)
0801         xmath = strrep(xmath, 'x', sprintf('%s', a.data.xunits));
0802         xlstr = [xlstr ' [' xmath '] '];
0803     else
0804         xlstr = [xlstr ' [' a.data.xunits '] '];
0805     end
0806     xlabel(xlstr);
0807

```



```

0808
0809 % Set grid on or off
0810 grid(tsax(end), 'on');
0811
0812 %----- line properties
0813
0814 % Set line color
0815 set(tqli(end), 'Color', col);
0816
0817 % Set specific colors
0818 if j<=length(linecolors) && ~isempty(linecolors{j})
0819     set(tqli(end), 'Color', linecolors{j});
0820 end
0821
0822 % Set line style
0823 if j<=length(linestyles) && ~isempty(linestyles{j})
0824     set(tqli(end), 'LineStyle', linestyles{j});
0825 end
0826 % Set markers
0827 if j<=length(markers) && ~isempty(markers{j})
0828     set(tqli(end), 'Marker', markers{j});
0829 end
0830
0831 % Set line widths
0832 if j<=length(linewidths) && ~isempty(linewidths(j))
0833     set(tqli(end), 'LineWidth', linewidths(j));
0834 end
0835
0836 % Set legend string
0837 lstr = '';
0838 if legendsOn
0839     if j<=length(legends) && ~isempty(legends{j})
0840         lstr = legends{j};
0841     else
0842         lstr = ltpda_label(a.name);
0843     end
0844 end
0845 legendStr = [legendStr cellstr(lstr)];
0846
0847 % Set the legend now if we can
0848 if legendsOn
0849     if strcmp(arrangement, 'single') || strcmp(arrangement, 'subplots')
0850         legend(legendStr{end});
0851     end
0852 end
0853 end
0854
0855 % Process legends for stacked plots
0856 if legendsOn
0857     if strcmp(arrangement, 'stacked')
0858         h = legend(legendStr);
0859         set(h, 'FontSize', 10)

```

```

0860     end
0861 end
0862
0863 end
0864
0865 % Set outputs
0866 if nargin > 0
0867     varargout{1} = tsfig;
0868 end
0869 if nargin > 1
0870     varargout{2} = tsax;
0871 end
0872 if nargin == 3
0873     varargout{3} = tsli;
0874 end
0875 if nargin > 3
0876     error('### Too many output arguments');
0877 end
0878
0879
0880 %-----
0881 % Plot cdata objects
0882 %
0883 function varargout = y_plot(varargin)
0884
0885 aos = varargin{1};
0886 pl  = varargin{2};
0887
0888 % Extract parameters
0889 arrangement = find(pl, 'Arrangement');
0890 linecolors  = find(pl, 'LineColors');
0891 colors      = find(pl, 'Colors');
0892 linestyles  = find(pl, 'LineStyles');
0893 markers     = find(pl, 'Markers');
0894 linewidths  = find(pl, 'Linewidths');
0895 legends     = find(pl, 'Legends');
0896 ylabel      = find(pl, 'YLabels');
0897 xlabel      = find(pl, 'XLabels');
0898 xmaths      = find(pl, 'XMaths');
0899 ymaths      = find(pl, 'YMaths');
0900
0901 % check whether we want legends or not
0902 if iscell(legends)
0903     legendsOn = 1;
0904 else
0905     if strcmp(legends, 'off')
0906         legendsOn = 0;
0907     else
0908         legendsOn = 1;
0909         legends = [];
0910     end

```

```

0911 end
0912
0913 if ~iscell(linestyles), linestyles = {linestyles}; end
0914 if ~iscell(linecolors), linecolors = {linecolors}; end
0915 if ~iscell(markers), markers = {markers}; end
0916 if ~iscell(legends), legends = {legends}; end
0917 if ~iscell(ylabels), ylabel = {ylabels}; end
0918 if ~iscell(xlabels), xlabel = {xlabel}; end
0919 if ~iscell(xmaths), xmaths = {xmaths}; end
0920 if ~iscell(ymaths), ymaths = {ymaths}; end
0921
0922 % collect figure handles
0923 tsfig = [];
0924 tsax = [];
0925 tsli = [];
0926
0927 % Legend holder
0928 legendStr = [];
0929
0930 if ~isempty(aos)
0931
0932     % Now loop over AOs
0933     Na = length(aos);
0934     for j=1:Na
0935
0936         % Get this AO
0937         a = aos(j);
0938
0939         % what figures do we need?
0940         switch arrangement
0941             case 'single'
0942
0943                 tsfig = [tsfig figure];
0944                 tsax = subplot(1,1,1);
0945                 col = colors{1};
0946
0947             case 'stacked'
0948
0949                 if j==1, tsfig = figure; end
0950                 tsax = subplot(1,1,1);
0951                 col = colors{mod(j-1,length(colors))+1};
0952                 hold on;
0953
0954             case 'subplots'
0955
0956                 if j == 1, tsfig = figure; end
0957                 tsax = [tsax subplot(Na, 1, j)];
0958                 col = colors{1};
0959
0960             otherwise
0961                 error('### Unknown plot arrangement');
0962         end
0963     end

```

```

0964 %----- Apply math functions
0965 x = 1:length(a.data.y);
0966 y = a.data.y;
0967
0968 ymath = '';
0969 if j<=length(ymaths)
0970     if ~isempty(ymaths{j})
0971         eval(sprintf('y = %s;', ymaths{j}));
0972         ymath = ymaths{j};
0973     end
0974 end
0975 xmath = '';
0976 if j<=length(xmaths)
0977     if ~isempty(xmaths{j})
0978         eval(sprintf('x = %s;', xmaths{j}));
0979         xmath = xmaths{j};
0980     end
0981 end
0982
0983 %----- Plot the data
0984
0985 idcs = plot(tsax(end), x, y);
0986 tsli = [tsli idcs(1:end).'];
0987
0988 %----- Axis properties
0989
0990 % Set ylabel
0991 if j<=length(ylabels) && ~isempty(ylabels{j})
0992     ylstr = ylabels{j};
0993 else
0994     ylstr = find(pl, 'YLabel');
0995 end
0996 if ~isempty(ymath)
0997     ymath = strrep(ymath, 'y', sprintf('%s', a.data.yunits));
0998     ylstr = [ylstr ' [' ymath '] '];
0999 else
1000     ylstr = [ylstr ' [' a.data.yunits '] '];
1001 end
1002 ylabel(ylstr);
1003
1004 % Set xlabel
1005 if j<=length(xlabels) && ~isempty(xlabels{j})
1006     xlstr = xlabels{j};
1007 else
1008     xlstr = find(pl, 'XLabel');
1009 end
1010 if ~isempty(xmath)
1011     xmath = strrep(xmath, 'x', sprintf('%s', a.data.xunits));
1012     xlstr = [xlstr ' [' xmath '] '];
1013 else
1014     xlstr = [xlstr ' [' a.data.xunits '] '];
1015 end

```

```

1016     xlabel(xlstr);
1017
1018
1019     % Set grid on or off
1020     grid(tsax(end), 'on');
1021
1022     %----- line properties
1023
1024     % Set line color
1025     set(tqli(end), 'Color', col);
1026
1027     % Override line colors with user defined colors
1028     if j<=length(linecolors) && ~isempty(linecolors{j})
1029         set(tqli(end), 'Color', linecolors{j});
1030     end
1031
1032     % Set line style
1033     if j<=length(linestyles) && ~isempty(linestyles{j})
1034         set(tqli(end), 'LineStyle', linestyles{j});
1035     end
1036
1037     % Set Markers
1038     if j<=length(markers) && ~isempty(markers{j})
1039         set(tqli(end), 'Marker', markers{j});
1040     end
1041
1042     % Set line widths
1043     if j<=length(linewidths)
1044         set(tqli(end), 'LineWidth', linewidths(j));
1045     end
1046
1047     % Set legend string
1048     if legendsOn
1049         if j<=length(legends) && ~isempty(legends{j})
1050             legendStr = [legendStr legends{j}];
1051         else
1052             lstr = ltpda_label(a.name);
1053             legendStr = [legendStr cellstr(lstr)];
1054         end
1055     end
1056
1057     % Set the legend now if we can
1058     if legendsOn
1059         if strcmp(arrangement, 'single') || strcmp(arrangement, 'subplots')
1060             legend(legendStr{end});
1061         end
1062     end
1063 end
1064
1065 % Process legends for stacked plots
1066 if legendsOn
1067     if strcmp(arrangement, 'stacked')

```

```

1068     h = legend(legendStr);
1069     set(h, 'FontSize', 10)
1070 end
1071 end
1072
1073 end
1074
1075 % Set outputs
1076 if nargin > 0
1077     varargout{1} = tsfig;
1078 end
1079 if nargin > 1
1080     varargout{2} = tsax;
1081 end
1082 if nargin == 3
1083     varargout{3} = tsli;
1084 end
1085 if nargin > 3
1086     error('### Too many output arguments');
1087 end
1088
1089 %-----
1090 % Plot xyzdata objects
1091 %
1092 function varargout = xyz_plot(varargin)
1093
1094 aos = varargin{1};
1095 pl  = varargin{2};
1096
1097 % Extract parameters
1098 arrangement = find(pl, 'Arrangement');
1099 linecolors   = find(pl, 'LineColors');
1100 colors       = find(pl, 'Colors');
1101 linestyles   = find(pl, 'LineStyles');
1102 linewidths   = find(pl, 'Linewidths');
1103 legends      = find(pl, 'Legends');
1104 zlabels      = find(pl, 'ZLabels');
1105 ylabels      = find(pl, 'YLabels');
1106 xlabels      = find(pl, 'XLabels');
1107 xmaths       = find(pl, 'XMaths');
1108 ymaths       = find(pl, 'YMaths');
1109 zmaths       = find(pl, 'ZMaths');
1110
1111 % check whether we want legends or not
1112 if iscell(legends)
1113     legendsOn = 1;
1114 else
1115     if strcmp(legends, 'off')
1116         legendsOn = 0;
1117     else
1118         legendsOn = 1;

```

```

1119     legends = [];
1120 end
1121 end
1122
1123 % collect figure handles
1124 tsfig = [];
1125 tsax  = [];
1126 tsli  = [];
1127
1128 % Legend holder
1129 legendStr = [];
1130
1131 if ~isempty(aos)
1132
1133     % Now loop over AOs
1134     Na = length(aos);
1135     for j=1:Na
1136
1137         % Get this AO
1138         a = aos(j);
1139
1140         % what figures do we need?
1141         switch arrangement
1142             case 'single'
1143
1144                 tsfig = [tsfig figure];
1145                 tsax = subplot(1,1,1);
1146
1147             case 'subplots'
1148
1149                 if j == 1, tsfig = figure; end
1150                 tsax = [tsax subplot(Na, 1, j)];
1151
1152             otherwise
1153                 error('### Unknown plot arrangement');
1154         end
1155
1156         %----- Apply math functions
1157         x = a.data.x;
1158         y = a.data.y;
1159         z = a.data.z;
1160
1161         ymath = '';
1162         if j<=length(ymaths)
1163             if ~isempty(ymaths{j})
1164                 eval(sprintf('y = %s;', ymaths{j}));
1165                 ymath = ymaths{j};
1166             end
1167         end
1168         xmath = '';
1169         if j<=length(xmaths)
1170             if ~isempty(xmaths{j})

```

```

1172         eval(sprintf('x = %s;', xmaths{j}));
1173         xmath = xmaths{j};
1174     end
1175 end
1176 zmath = '';
1177 if j<=length(zmaths)
1178     if ~isempty(zmaths{j})
1179         eval(sprintf('z = %s;', zmaths{j}));
1180         zmath = zmaths{j};
1181     end
1182 end
1183
1184 %----- Plot the data
1185
1186 idcs = pcolor(x,y,z);
1187 tsli = [tsli idcs(1:end).'];
1188
1189 % plot properties
1190 set(idcs, 'EdgeColor', 'none');
1191
1192 %----- Axis properties
1193
1194 % Reverse y-direction for spectrograms
1195 set(tsax(end), 'YDir', 'reverse');
1196
1197 % Set ylabel
1198 if j<=length(ylabels) && ~isempty(ylabels{j})
1199     ylstr = ylabels{j};
1200 else
1201     ylstr = find(pl, 'YLabel');
1202 end
1203 if ~isempty(ymath)
1204     ymath = strrep(ymath, 'y', sprintf('%s', a.data.yunits));
1205     ylstr = [ylstr ' [' ymath '] '];
1206 else
1207     ylstr = [ylstr ' [' a.data.yunits '] '];
1208 end
1209 ylabel(ylstr);
1210
1211 % Set xlabel
1212 if j<=length(xlabels) && ~isempty(xlabels{j})
1213     xlstr = xlabels{j};
1214 else
1215     xlstr = find(pl, 'XLabel');
1216 end
1217 if ~isempty(xmath)
1218     xmath = strrep(xmath, 'x', sprintf('%s', a.data.xunits));
1219     xlstr = [xlstr ' [' xmath '] '];
1220 else
1221     xlstr = [xlstr ' [' a.data.xunits '] '];
1222 end
1223 xlabel(xlstr);

```



```

1224
1225     % Set grid on or off
1226     grid(tsax(end), 'on');
1227
1228     % Set title string
1229     if ~strcmpi(legends, 'off')
1230         if j<=length(legends) && ~isempty(legends{j})
1231             legendStr = [legendStr legends(j)];
1232         else
1233             lstr = ltpda_label(a.name);
1234             legendStr = [legendStr cellstr(lstr)];
1235         end
1236     end
1237
1238     % Set the legend now if we can
1239     tstr = legendStr{end};
1240     if legendsOn
1241         title(tstr);
1242     end
1243
1244     % Set colorbars
1245     hc = colorbar('peer', tsax(end));
1246     if j<=length(zlabels)
1247         if ~isempty(zlabels{j})
1248             zlstr = zlabels{j};
1249         end
1250     else
1251         zlstr = find(pl, 'Zlabel');
1252     end
1253     if ~isempty(zmath), zlstr = [zlstr sprintf('\n%s', zmath)]; end
1254     ylh = get(hc, 'YLabel');
1255     set(ylh, 'String', zlstr);
1256     set(ylh, 'FontSize', get(tsax(end), 'FontSize'))
1257     set(ylh, 'FontName', get(tsax(end), 'FontName'))
1258     set(ylh, 'FontAngle', get(tsax(end), 'FontAngle'))
1259     set(ylh, 'FontWeight', get(tsax(end), 'FontWeight'))
1260 end
1261 end
1262
1263 % Set outputs
1264 if nargsout > 0
1265     varargout{1} = tsfig;
1266 end
1267 if nargsout > 1
1268     varargout{2} = tsax;
1269 end
1270 if nargsout == 3
1271     varargout{3} = tsli;
1272 end
1273 if nargsout > 3
1274     error('### Too many output arguments');
1275 end

```

```

1276
1277
1278 %-----
1279 % Default Parameter Lists
1280 %
1281 function out = getDefaultPlist(varargin)
1282
1283 % list of available parameter sets
1284 sets = {'tsdata', 'fsdata', 'cdata', 'xydata'};
1285
1286 % Get the LTPDA color set for lines
1287 colors = getappdata(0, 'ltpda_default_plot_colors');
1288
1289 if nargin == 0
1290     out = sets;
1291     return
1292 end
1293
1294 set = varargin{1};
1295
1296 out = plist('Colors', colors, ...
1297     'Arrangement', 'stacked');
1298
1299 switch set
1300 case 'fsdata'
1301     out = append(out, 'type', 'fsdata', ...
1302         'complexPlotType', 'absdeg', ...
1303         'XLabel', 'Frequency');
1304 case 'tsdata'
1305     out = append(out, 'type', 'tsdata', ...
1306         'Xlabel', 'Time', ...
1307         'Ylabel', 'Amplitude');
1308 case 'xydata'
1309     out = append(out, 'type', 'xydata', ...
1310         'Xlabel', 'X-data', ...
1311         'Ylabel', 'Y-data', ...
1312         'YMaths', '', ...
1313         'XMaths', '');
1314 case 'xyzdata'
1315     out = plist('Colors', colors, 'Arrangement', 'single', ...
1316         'type', 'xydata', ...
1317         'Xlabel', 'Time', ...
1318         'Ylabel', 'Frequency', ...
1319         'Zlabel', 'Amplitude', ...
1320         'YMaths', '', ...
1321         'ZMaths', '', ...
1322         'XMaths', '');
1323 case 'cdata'
1324     out = append(out, 'type', 'tsdata', ...
1325         'Xlabel', 'Index', ...
1326         'Ylabel', 'Value');
1327 otherwise

```

```

1328     out = plist();
1329 end
1330
1331
1332
1333
1334 %% OLD BELOW HERE
1335 % function varargout = iplot(varargin)
1336 % % PLOT plots an array of analysis objects.
1337 % %
1338 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1339 % %
1340 % % DESCRIPTION: PLOT plots an array of analysis objects.
1341 % %
1342 % % CALL:
1343 % %
1344 % % PARAMETERS:  complex_view    're/im'    'mag/deg'
1345 % %               title
1346 % %               enbw
1347 % %               legend_names
1348 % %
1349 % % VERSION:      $Id: iplot.m,v 1.17 2008/01/21 11:40:25 hewitson Exp $
1350 % %
1351 % % The following call returns a parameter list object that contains the
1352 % % default parameter values:
1353 % %
1354 % % >> pl = plot(ao, 'Params')
1355 % %
1356 % % HISTORY: 21-08-2007 Diepholz
1357 % %           Creation
1358 % %
1359 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1360 %
1361 % VERSION = '$Id: iplot.m,v 1.17 2008/01/21 11:40:25 hewitson Exp $';
1362 %
1363 % %%%%%%%%% Check if this is a call for parameters %%%%%%%%%
1364 % if nargin == 2
1365 %     if isa(varargin{1}, 'ao') && ischar(varargin{2})
1366 %         in = char(varargin{2});
1367 %         if strcmp(in, 'Params')
1368 %             varargout{1} = getDefaultPL();
1369 %         elseif strcmp(in, 'Version')
1370 %             varargout{1} = VERSION;
1371 %             return
1372 %         end
1373 %     end
1374 % end
1375 %
1376 % axes_handle = [];
1377 % ao_s        = [];
1378 % pl_s        = [];

```

```

1379 %     line_spec      = [];
1380 %     prop_val       = {};
1381 %
1382 %     %%%%%%%%% Check input values %%%%%%%%%
1383 %     var_argin = varargin;
1384 %
1385 %     while ~isempty(var_argin)
1386 %         if ishandle(var_argin{1}) & isempty(axes_handle)
1387 %             axes_handle = var_argin{1};
1388 %             var_argin = var_argin(2:end);
1389 %         elseif isa(var_argin{1}, 'ao')
1390 %             var_argin{1} = reshape(var_argin{1}, 1, []);
1391 %             ao_s = [ao_s var_argin{1}];
1392 %             var_argin = var_argin(2:end);
1393 %         elseif isa(var_argin{1}, 'plist')
1394 %             pl_s = [pl_s var_argin{1}];
1395 %             var_argin = var_argin(2:end);
1396 %         elseif ischar(var_argin{1}) && islinespec(var_argin{1}) && isempty
1397 % (line_spec)
1398 %             line_spec = var_argin{1};
1399 %             var_argin = var_argin(2:end);
1400 %         else
1401 %             if length(var_argin) == 1
1402 %                 error('### There is only one property [%s] without value left',
1403 % var_argin{1});
1404 %             elseif length(var_argin) >= 2
1405 %                 prop = var_argin{1};
1406 %                 val  = var_argin{2};
1407 %                 if ismember(prop, {'complex_view', ...
1408 %                                     'title_str', ...
1409 %                                     'legend_names', ...
1410 %                                     'xlabel_des', ...
1411 %                                     'ylabel_des'})
1412 %                     pl_s = [pl_s plist(prop, val)];
1413 %                 else
1414 %                     prop_val = [prop_val, prop, val];
1415 %                 end
1416 %                 var_argin = var_argin(3:end);
1417 %             end
1418 %         end
1419 %     end
1420 %
1421 %     pl_s = combine(pl_s, getDefaultPL());
1422 %
1423 %     %%%%%%%%% Get default values from the application-defined data. %%%%%%%%%
1424 %
1425 %     len_legend = getappdata(0, 'wraplegendstringat');
1426 %
1427 %     %%%%%%%%% Check some display propeties. %%%%%%%%%
1428 %
1429 %     sort_ao = [];

```

```

1429 %   T0_off  = 1e20;
1430 %
1431 %   for ii = 1:length(ao_s)
1432 %
1433 %       [x,y] = get_xy_values(ao_s(ii).data);
1434 %
1435 %       %%%%%% Get the class name
1436 %       class_name = class(ao_s(ii).data);
1437 %
1438 %       % If the class_name doesn't exist in the variable sort_ao then add this
name
1439 %       % as a struct.
1440 %       if isempty(sort_ao) || ~ismember(class_name, fieldnames(sort_ao))
1441 %           sort_ao.(class_name) = [];
1442 %       end
1443 %
1444 %       %%%%%% Define the necessary fields (default values) %%%%%%
1445 %       if isempty(sort_ao.(class_name))
1446 %           sort_ao.(class_name) = struct('ao',          [],          ...
1447 %                                         'complex',      false,      ...
1448 %                                         'complex_view', '',          ...
1449 %                                         'title_str',     '',          ...
1450 %                                         'legend_names',  '',          ...
1451 %                                         'xlabel_des',    '',          ...
1452 %                                         'ylabel_des',    '',          ...
1453 %                                         'xscale',        'linear',    ...
1454 %                                         'yscale',        'linear',    ...
1455 %                                         'enbw',          false);
1456 %       end
1457 %
1458 %       %%%%%% Set the fields for the associated data %%%%%%
1459 %       sort_ao.(class_name).ao = [sort_ao.(class_name).ao ao_s(ii)];
1460 %       sort_ao.(class_name).complex_view = find(pl_s, 'complex_view');
1461 %       sort_ao.(class_name).title_str    = find(pl_s, 'title_str');
1462 %       sort_ao.(class_name).legend_names = find(pl_s, 'legend_names');
1463 %       sort_ao.(class_name).xlabel_des   = find(pl_s, 'xlabel_des');
1464 %       sort_ao.(class_name).ylabel_des   = find(pl_s, 'ylabel_des');
1465 %
1466 %       if ~isreal(y)
1467 %           sort_ao.(class_name).complex = true;
1468 %       end
1469 %       if (max(x)-min(x)) / 10^floor(log10(mean(abs(x)))) > 9
1470 %           sort_ao.(class_name).xscale = 'log';
1471 %       end
1472 %       if min(x) < 0
1473 %           sort_ao.(class_name).xscale = 'linear';
1474 %       end
1475 %       if (max(y)-min(y)) / 10^floor(log10(max(mean(abs(y))))) > 9
1476 %           sort_ao.(class_name).yscale = 'log';
1477 %       end
1478 %       if min(y) < 0
1479 %           sort_ao.(class_name).yscale = 'linear';
1480 %       end

```

```

1481 %         if strcmp(class_name, 'tsdata')
1482 %             sort_ao.(class_name).xscale = 'linear';
1483 %         end
1484 %
1485 %
1486 %         %%%%%%%%%%%%% Spezial fields for the different data objects %%%%%%%%%%%%%%%
1487 %
1488 %         %%%%%%%%% FSDATA %%%%%%%%%
1489 %         if strcmp(class_name, 'fsdata')
1490 %             if length (ao_s(ii).data.enbw) > 1
1491 %                 sort_ao.(class_name).enbw = find(pl_s, 'enbw');
1492 %             end
1493 %
1494 %             % For fsdata set the x-axis and the y-axis to a logarithmic scale
1495 %             sort_ao.(class_name).xscale = 'log';
1496 %             sort_ao.(class_name).yscale = 'log';
1497 %         end
1498 %
1499 %         %%%%%%%%% TSDATA %%%%%%%%%
1500 %         if strcmp(class_name, 'tsdata')
1501 %
1502 %             t0 = ao_s(ii).data.t0.utc_epoch_milli/1000;
1503 %             if t0 < T0_off
1504 %                 T0_off = t0;
1505 %                 sort_ao.(class_name).T0_off = t0;
1506 %             end
1507 %         end
1508 %
1509 %     end
1510 %
1511 %     %%%%%%%%%%%%%%%
1512 %     %% Plot the different data objects %%
1513 %     %%%%%%%%%%%%%%%
1514 %     fields = fieldnames(sort_ao);
1515 %
1516 %     out_line_h    = [];
1517 %     out_axes_h    = [];
1518 %     out_figure_h  = [];
1519 %
1520 %     for ii = 1:length(fields)
1521 %         class_name = fields{ii};
1522 %
1523 %         %%%%%%%%%%%%% Add the t0 offset to all aos with tsdata objects. %%%%%%%%%%%%%%%
1524 %         %
1525 %         if strcmp(class_name, 'tsdata')
1526 %             for jj = 1:length(sort_ao.(class_name).ao)
1527 %                 curr_ao = sort_ao.(class_name).ao(jj);
1528 %
1529 %                 t0 = curr_ao.data.t0.utc_epoch_milli/1000;
1530 %                 curr_ao.data.x = curr_ao.data.x + t0 - sort_ao.(class_name).T0_off;
1531 %
1532 %                 sort_ao.(class_name).ao(jj) = curr_ao;
1533 %             end
1534 %         end
1535 %     end
1536 %
1537 %     %%%%%%%%%%%%%%%
1538 %     %% Plot the different data objects %%
1539 %     %%%%%%%%%%%%%%%
1540 %
1541 %     %%%%%%%%%%%%%%%
1542 %     %% Plot the different data objects %%
1543 %     %%%%%%%%%%%%%%%
1544 %
1545 %     %%%%%%%%%%%%%%%
1546 %     %% Plot the different data objects %%
1547 %     %%%%%%%%%%%%%%%
1548 %
1549 %     %%%%%%%%%%%%%%%
1550 %     %% Plot the different data objects %%
1551 %     %%%%%%%%%%%%%%%
1552 %
1553 %     %%%%%%%%%%%%%%%
1554 %     %% Plot the different data objects %%
1555 %     %%%%%%%%%%%%%%%
1556 %
1557 %     %%%%%%%%%%%%%%%
1558 %     %% Plot the different data objects %%
1559 %     %%%%%%%%%%%%%%%
1560 %
1561 %     %%%%%%%%%%%%%%%
1562 %     %% Plot the different data objects %%
1563 %     %%%%%%%%%%%%%%%
1564 %
1565 %     %%%%%%%%%%%%%%%
1566 %     %% Plot the different data objects %%
1567 %     %%%%%%%%%%%%%%%
1568 %
1569 %     %%%%%%%%%%%%%%%
1570 %     %% Plot the different data objects %%
1571 %     %%%%%%%%%%%%%%%
1572 %
1573 %     %%%%%%%%%%%%%%%
1574 %     %% Plot the different data objects %%
1575 %     %%%%%%%%%%%%%%%
1576 %
1577 %     %%%%%%%%%%%%%%%
1578 %     %% Plot the different data objects %%
1579 %     %%%%%%%%%%%%%%%
1580 %
1581 %     %%%%%%%%%%%%%%%
1582 %     %% Plot the different data objects %%
1583 %     %%%%%%%%%%%%%%%
1584 %
1585 %     %%%%%%%%%%%%%%%
1586 %     %% Plot the different data objects %%
1587 %     %%%%%%%%%%%%%%%
1588 %
1589 %     %%%%%%%%%%%%%%%
1590 %     %% Plot the different data objects %%
1591 %     %%%%%%%%%%%%%%%
1592 %
1593 %     %%%%%%%%%%%%%%%
1594 %     %% Plot the different data objects %%
1595 %     %%%%%%%%%%%%%%%
1596 %
1597 %     %%%%%%%%%%%%%%%
1598 %     %% Plot the different data objects %%
1599 %     %%%%%%%%%%%%%%%
1600 %
1601 %     %%%%%%%%%%%%%%%
1602 %     %% Plot the different data objects %%
1603 %     %%%%%%%%%%%%%%%
1604 %
1605 %     %%%%%%%%%%%%%%%
1606 %     %% Plot the different data objects %%
1607 %     %%%%%%%%%%%%%%%
1608 %
1609 %     %%%%%%%%%%%%%%%
1610 %     %% Plot the different data objects %%
1611 %     %%%%%%%%%%%%%%%
1612 %
1613 %     %%%%%%%%%%%%%%%
1614 %     %% Plot the different data objects %%
1615 %     %%%%%%%%%%%%%%%
1616 %
1617 %     %%%%%%%%%%%%%%%
1618 %     %% Plot the different data objects %%
1619 %     %%%%%%%%%%%%%%%
1620 %
1621 %     %%%%%%%%%%%%%%%
1622 %     %% Plot the different data objects %%
1623 %     %%%%%%%%%%%%%%%
1624 %
1625 %     %%%%%%%%%%%%%%%
1626 %     %% Plot the different data objects %%
1627 %     %%%%%%%%%%%%%%%
1628 %
1629 %     %%%%%%%%%%%%%%%
1630 %     %% Plot the different data objects %%
1631 %     %%%%%%%%%%%%%%%
1632 %
1633 %     %%%%%%%%%%%%%%%
1634 %     %% Plot the different data objects %%
1635 %     %%%%%%%%%%%%%%%
1636 %
1637 %     %%%%%%%%%%%%%%%
1638 %     %% Plot the different data objects %%
1639 %     %%%%%%%%%%%%%%%
1640 %
1641 %     %%%%%%%%%%%%%%%
1642 %     %% Plot the different data objects %%
1643 %     %%%%%%%%%%%%%%%
1644 %
1645 %     %%%%%%%%%%%%%%%
1646 %     %% Plot the different data objects %%
1647 %     %%%%%%%%%%%%%%%
1648 %
1649 %     %%%%%%%%%%%%%%%
1650 %     %% Plot the different data objects %%
1651 %     %%%%%%%%%%%%%%%
1652 %
1653 %     %%%%%%%%%%%%%%%
1654 %     %% Plot the different data objects %%
1655 %     %%%%%%%%%%%%%%%
1656 %
1657 %     %%%%%%%%%%%%%%%
1658 %     %% Plot the different data objects %%
1659 %     %%%%%%%%%%%%%%%
1660 %
1661 %     %%%%%%%%%%%%%%%
1662 %     %% Plot the different data objects %%
1663 %     %%%%%%%%%%%%%%%
1664 %
1665 %     %%%%%%%%%%%%%%%
1666 %     %% Plot the different data objects %%
1667 %     %%%%%%%%%%%%%%%
1668 %
1669 %     %%%%%%%%%%%%%%%
1670 %     %% Plot the different data objects %%
1671 %     %%%%%%%%%%%%%%%
1672 %
1673 %     %%%%%%%%%%%%%%%
1674 %     %% Plot the different data objects %%
1675 %     %%%%%%%%%%%%%%%
1676 %
1677 %     %%%%%%%%%%%%%%%
1678 %     %% Plot the different data objects %%
1679 %     %%%%%%%%%%%%%%%
1680 %
1681 %     %%%%%%%%%%%%%%%
1682 %     %% Plot the different data objects %%
1683 %     %%%%%%%%%%%%%%%
1684 %
1685 %     %%%%%%%%%%%%%%%
1686 %     %% Plot the different data objects %%
1687 %     %%%%%%%%%%%%%%%
1688 %
1689 %     %%%%%%%%%%%%%%%
1690 %     %% Plot the different data objects %%
1691 %     %%%%%%%%%%%%%%%
1692 %
1693 %     %%%%%%%%%%%%%%%
1694 %     %% Plot the different data objects %%
1695 %     %%%%%%%%%%%%%%%
1696 %
1697 %     %%%%%%%%%%%%%%%
1698 %     %% Plot the different data objects %%
1699 %     %%%%%%%%%%%%%%%
1700 %
1701 %     %%%%%%%%%%%%%%%
1702 %     %% Plot the different data objects %%
1703 %     %%%%%%%%%%%%%%%
1704 %
1705 %     %%%%%%%%%%%%%%%
1706 %     %% Plot the different data objects %%
1707 %     %%%%%%%%%%%%%%%
1708 %
1709 %     %%%%%%%%%%%%%%%
1710 %     %% Plot the different data objects %%
1711 %     %%%%%%%%%%%%%%%
1712 %
1713 %     %%%%%%%%%%%%%%%
1714 %     %% Plot the different data objects %%
1715 %     %%%%%%%%%%%%%%%
1716 %
1717 %     %%%%%%%%%%%%%%%
1718 %     %% Plot the different data objects %%
1719 %     %%%%%%%%%%%%%%%
1720 %
1721 %     %%%%%%%%%%%%%%%
1722 %     %% Plot the different data objects %%
1723 %     %%%%%%%%%%%%%%%
1724 %
1725 %     %%%%%%%%%%%%%%%
1726 %     %% Plot the different data objects %%
1727 %     %%%%%%%%%%%%%%%
1728 %
1729 %     %%%%%%%%%%%%%%%
1730 %     %% Plot the different data objects %%
1731 %     %%%%%%%%%%%%%%%
1732 %
1733 %     %%%%%%%%%%%%%%%
1734 %     %% Plot the different data objects %%
1735 %     %%%%%%%%%%%%%%%
1736 %
1737 %     %%%%%%%%%%%%%%%
1738 %     %% Plot the different data objects %%
1739 %     %%%%%%%%%%%%%%%
1740 %
1741 %     %%%%%%%%%%%%%%%
1742 %     %% Plot the different data objects %%
1743 %     %%%%%%%%%%%%%%%
1744 %
1745 %     %%%%%%%%%%%%%%%
1746 %     %% Plot the different data objects %%
1747 %     %%%%%%%%%%%%%%%
1748 %
1749 %     %%%%%%%%%%%%%%%
1750 %     %% Plot the different data objects %%
1751 %     %%%%%%%%%%%%%%%
1752 %
1753 %     %%%%%%%%%%%%%%%
1754 %     %% Plot the different data objects %%
1755 %     %%%%%%%%%%%%%%%
1756 %
1757 %     %%%%%%%%%%%%%%%
1758 %     %% Plot the different data objects %%
1759 %     %%%%%%%%%%%%%%%
1760 %
1761 %     %%%%%%%%%%%%%%%
1762 %     %% Plot the different data objects %%
1763 %     %%%%%%%%%%%%%%%
1764 %
1765 %     %%%%%%%%%%%%%%%
1766 %     %% Plot the different data objects %%
1767 %     %%%%%%%%%%%%%%%
1768 %
1769 %     %%%%%%%%%%%%%%%
1770 %     %% Plot the different data objects %%
1771 %     %%%%%%%%%%%%%%%
1772 %
1773 %     %%%%%%%%%%%%%%%
1774 %     %% Plot the different data objects %%
1775 %     %%%%%%%%%%%%%%%
1776 %
1777 %     %%%%%%%%%%%%%%%
1778 %     %% Plot the different data objects %%
1779 %     %%%%%%%%%%%%%%%
1780 %
1781 %     %%%%%%%%%%%%%%%
1782 %     %% Plot the different data objects %%
1783 %     %%%%%%%%%%%%%%%
1784 %
1785 %     %%%%%%%%%%%%%%%
1786 %     %% Plot the different data objects %%
1787 %     %%%%%%%%%%%%%%%
1788 %
1789 %     %%%%%%%%%%%%%%%
1790 %     %% Plot the different data objects %%
1791 %     %%%%%%%%%%%%%%%
1792 %
1793 %     %%%%%%%%%%%%%%%
1794 %     %% Plot the different data objects %%
1795 %     %%%%%%%%%%%%%%%
1796 %
1797 %     %%%%%%%%%%%%%%%
1798 %     %% Plot the different data objects %%
1799 %     %%%%%%%%%%%%%%%
1800 %
1801 %     %%%%%%%%%%%%%%%
1802 %     %% Plot the different data objects %%
1803 %     %%%%%%%%%%%%%%%
1804 %
1805 %     %%%%%%%%%%%%%%%
1806 %     %% Plot the different data objects %%
1807 %     %%%%%%%%%%%%%%%
1808 %
1809 %     %%%%%%%%%%%%%%%
1810 %     %% Plot the different data objects %%
1811 %     %%%%%%%%%%%%%%%
1812 %
1813 %     %%%%%%%%%%%%%%%
1814 %     %% Plot the different data objects %%
1815 %     %%%%%%%%%%%%%%%
1816 %
1817 %     %%%%%%%%%%%%%%%
1818 %     %% Plot the different data objects %%
1819 %     %%%%%%%%%%%%%%%
1820 %
1821 %     %%%%%%%%%%%%%%%
1822 %     %% Plot the different data objects %%
1823 %     %%%%%%%%%%%%%%%
1824 %
1825 %     %%%%%%%%%%%%%%%
1826 %     %% Plot the different data objects %%
1827 %     %%%%%%%%%%%%%%%
1828 %
1829 %     %%%%%%%%%%%%%%%
1830 %     %% Plot the different data objects %%
1831 %     %%%%%%%%%%%%%%%
1832 %
1833 %     %%%%%%%%%%%%%%%
1834 %     %% Plot the different data objects %%
1835 %     %%%%%%%%%%%%%%%
1836 %
1837 %     %%%%%%%%%%%%%%%
1838 %     %% Plot the different data objects %%
1839 %     %%%%%%%%%%%%%%%
1840 %
1841 %     %%%%%%%%%%%%%%%
1842 %     %% Plot the different data objects %%
1843 %     %%%%%%%%%%%%%%%
1844 %
1845 %     %%%%%%%%%%%%%%%
1846 %     %% Plot the different data objects %%
1847 %     %%%%%%%%%%%%%%%
1848 %
1849 %     %%%%%%%%%%%%%%%
1850 %     %% Plot the different data objects %%
1851 %     %%%%%%%%%%%%%%%
1852 %
1853 %     %%%%%%%%%%%%%%%
1854 %     %% Plot
```

```

1533 %      end
1534 %
1535 %      %%%%%%%%% Set default values %%%%%%%%%
1536 %      % Set default title
1537 %      if isempty (sort_ao.(class_name).title_str)
1538 %          if sort_ao.(class_name).complex == true
1539 %              add_text = 'complex ';
1540 %          else
1541 %              add_text = '';
1542 %          end
1543 %          switch (class_name)
1544 %              case 'fsdata'
1545 %                  sort_ao.(class_name).title_str = sprintf('Plot of %sfrequency
series AOs', add_text);
1546 %              case 'tsdata'
1547 %                  sort_ao.(class_name).title_str = sprintf('Plot of %stime series
AOs', add_text);
1548 %              case 'cdata'
1549 %                  sort_ao.(class_name).title_str = sprintf('Plot of %sconstant data
AOs', add_text);
1550 %              case 'xydata'
1551 %                  sort_ao.(class_name).title_str = sprintf('Plot of %sxy data AOs',
add_text);
1552 %              otherwise
1553 %                  error('### Unknown data object ''%s'', class_name);
1554 %              end
1555 %          end
1556 %
1557 %      % Set default x-, y-units
1558 %      if isempty(sort_ao.(class_name).ao(1).data.xunits)
1559 %          xunits = 'unknown';
1560 %      else
1561 %          xunits= sort_ao.(class_name).ao(1).data.xunits;
1562 %      end
1563 %      if isempty(sort_ao.(class_name).ao(1).data.yunits)
1564 %          yunits = 'unknown';
1565 %      else
1566 %          yunits= sort_ao.(class_name).ao(1).data.yunits;
1567 %      end
1568 %
1569 %      fig_h = gcf;
1570 %
1571 %      %%%%%%%%% Plot complex values %%%%%%%%%
1572 %      if sort_ao.(class_name).complex == true
1573 %
1574 %          if strcmp(sort_ao.(class_name).complex_view, 're/im')
1575 %              axes_h1 = subplot(2,1,1);
1576 %              line_h1 = plot(axes_h1, real(sort_ao.(class_name).ao), line_spec,
prop_val{:});
1577 %              axes_h2 = subplot(2,1,2);
1578 %              line_h2 = plot(axes_h2, imag(sort_ao.(class_name).ao), line_spec,
prop_val{:});
1579 %

```



```

1580 %     label_x1 = [sort_ao.(class_name).xlabel_des ' [' xunits ']];
1581 %     label_y1 = [sort_ao.(class_name).ylabel_des 'Real [' yunits ']];
1582 %     label_y2 = [sort_ao.(class_name).ylabel_des 'Imaginary [' yunits ']];
1583 %
1584 %     % Do not set the xscale if it is set as a propertie/value pair
1585 %     if ~ismember('xscale', lower(prop_val))
1586 %         % If the 'key' exist in the parameter list then do NOT replace the
1587 %         % value. (see combine) If the user use a different syntax (he use
1588 %         % upper case letters) this value is put before the user value.
1589 %         % This function take ALWAYS the user value.
1590 %         pl_s = combine(pl_s, plist('xscale', sort_ao.(class_name).xscale));
1591 %     end
1592 %
1593 %     % Plot both y-axes as a linear scale
1594 %     pl_s = pset(pl_s, 'Yscale', 'linear');
1595 %
1596 % else % if strcmp(sort_ao.(class_name).complex_view, 'mag/deg')
1597 %     axes_h1 = subplot(3,1,1:2);
1598 %     line_h1 = plot(axes_h1, abs(sort_ao.(class_name).ao), line_spec,
prop_val{:});
1599 %     axes_h2 = subplot(3,1,3);
1600 %     line_h2 = plot(axes_h2, phase(sort_ao.(class_name).ao), line_spec,
prop_val{:});
1601 %
1602 %     label_x1 = [sort_ao.(class_name).xlabel_des ' [' xunits ']];
1603 %     label_y1 = [sort_ao.(class_name).ylabel_des 'Magnitude [' yunits ']];
1604 %     label_y2 = [sort_ao.(class_name).ylabel_des 'Phase [' yunits ']];
1605 %
1606 %     % Do not set the xscale if it is set as a propertie/value pair
1607 %     if ~ismember('xscale', lower(prop_val))
1608 %         % If the 'key' exist in the parameter list then do NOT replace the
1609 %         % value. (see combine) If the user use a different syntax (he use
1610 %         % upper case letters) this value is put before the user value.
1611 %         % This function take ALWAYS the user value.
1612 %         pl_s = combine(pl_s, plist('xscale', sort_ao.(class_name).xscale));
1613 %     end
1614 %
1615 %     % Do not set the yscale if it is set as a propertie/value pair
1616 %     if ~ismember('yscale', lower(prop_val))
1617 %         % If the 'key' exist in the parameter list then do NOT replace the
1618 %         % value. (see combine) If the user use a different syntax (he use
1619 %         % upper case letters) this value is put before the user value.
1620 %         % This function take ALWAYS the user value.
1621 %         pl_s = combine(pl_s, plist('yscale', sort_ao.(class_name).yscale));
1622 %     end
1623 %
1624 %     % Plot always the second y-axes as a linear scale
1625 %     pl_s = pset(pl_s, 'Yscale2', 'linear');
1626 % end
1627 %
1628 % % Set x-, y-label
1629 % xlabel(axes_h2, label_x1);
1630 % ylabel(axes_h1, label_y1);

```



```

1631 %         ylabel(axes_h2, label_y2);
1632 %
1633 %         out_line_h    = [out_line_h,    line_h1, line_h2];
1634 %         out_axes_h    = [out_axes_h,    axes_h1, axes_h2];
1635 %         out_figure_h = [out_figure_h, fig_h];
1636 %
1637 %         %%%%%%%%%%% Plot real values %%%%%%%%%%%
1638 %         else
1639 %
1640 %             [line_h1, axes_h1] = plot(real(sort_ao.(class_name).ao), line_spec,
prop_val{:});
1641 %
1642 %             axes_h2 = axes_h1;
1643 %             line_h2 = line_h1;
1644 %
1645 %             % Set x-, y-label
1646 %             xlabel(axes_h1, [sort_ao.(class_name).xlabel_des ' [' xunits ']']);
1647 %             ylabel(axes_h1, [sort_ao.(class_name).ylabel_des ' [' yunits ']']);
1648 %
1649 %             out_line_h    = [out_line_h,    line_h1];
1650 %             out_axes_h    = [out_axes_h,    axes_h1];
1651 %             out_figure_h = [out_figure_h, fig_h];
1652 %
1653 %             % Do not set the xscale if it is set as a propertie/value pair
1654 %             if ~ismember('xscale', lower(prop_val))
1655 %                 % If the 'key' exist in the parameter list then do NOT replace the
1656 %                 % value. (see combine) If the user use a different syntax (he use
1657 %                 % upper case letters) this value is put before the user value.
1658 %                 % This function take ALWAYS the user value.
1659 %                 pl_s = combine(pl_s, plist('xscale', sort_ao.(class_name).xscale));
1660 %             end
1661 %
1662 %             % Do not set the yscale if it is set as a propertie/value pair
1663 %             if ~ismember('yscale', lower(prop_val))
1664 %                 % If the 'key' exist in the parameter list then do NOT replace the
1665 %                 % value. (see combine) If the user use a different syntax (he use
1666 %                 % upper case letters) this value is put before the user value.
1667 %                 % This function take ALWAYS the user value.
1668 %                 pl_s = combine(pl_s, plist('yscale', sort_ao.(class_name).yscale));
1669 %             end
1670 %
1671 %         end % if complex == true
1672 %
1673 %         %%%%%%%%%%% Set Title %%%%%%%%%%%
1674 %         title (axes_h1, sort_ao.(class_name).title_str);
1675 %
1676 %         %%%%%%%%%%% Set Title %%%%%%%%%%%
1677 %         grid(axes_h1, 'on');
1678 %         grid(axes_h2, 'on');
1679 %
1680 %         %%%%%%%%%%% Set legend names %%%%%%%%%%%
1681 %         % Use the ao names for the legend
1682 %         if isempty(sort_ao.(class_name).legend_names)

```

```

1683 %         for jj = 1:length(sort_ao.(class_name).ao)
1684 %
1685 %             display_name = sort_ao.(class_name).ao(jj).name;
1686 %             display_name = ltpda_label(display_name);
1687 %             display_name = wrapLegendString(sprintf('%s', ltpda_label
(display_name)), len_legend);
1688 %             set(line_h1(jj), 'DisplayName', ltpda_label(display_name))
1689 %         end
1690 %         legend(axes_h1, 'toggle')
1691 %         % Use the legend names from the plist
1692 %     else
1693 %         legend(axes_h1, sort_ao.(class_name).legend_names);
1694 %     end
1695 %
1696 %     %%%%%%%%%%% Set axes OR lineseries properties from the plist %%%%%%%%%%%
1697 %     for jj = 1:length(pl_s.params)
1698 %         p = pl_s.params(jj);
1699 %
1700 %         [idx_b, idx_e, dummy, match] = regexp(p.key, '\d', 'once');
1701 %         p.key = strrep(p.key, match, '');
1702 %
1703 %         % Check key to the axes property
1704 %         if isprop(line_h1, p.key)
1705 %             if strcmp(match, '1')
1706 %                 set(line_h1, p.key, p.val);
1707 %             elseif strcmp(match, '2')
1708 %                 set(line_h2, p.key, p.val);
1709 %             else
1710 %                 set(line_h1, p.key, p.val);
1711 %                 set(line_h2, p.key, p.val);
1712 %             end
1713 %         % Check key to the axes property
1714 %         elseif isprop(axes_h1, p.key)
1715 %             if strcmp(match, '1')
1716 %                 set(axes_h1, p.key, p.val);
1717 %             elseif strcmp(match, '2')
1718 %                 set(axes_h2, p.key, p.val);
1719 %             else
1720 %                 set(axes_h1, p.key, p.val);
1721 %                 set(axes_h2, p.key, p.val);
1722 %             end
1723 %         end
1724 %     end
1725 % end
1726 %
1727 % end
1728 %
1729 %
1730 % %%%%%%%%%%% varargout{1} = line      handle %%%%%%%%%%%
1731 % %%%%%%%%%%% varargout{2} = axes      handle %%%%%%%%%%%
1732 % %%%%%%%%%%% varargout{3} = figure    handle %%%%%%%%%%%
1733 %
1734 % if nargin == 0

```

```

1735 %     elseif nargsout == 1
1736 %         varargout{1} = out_line_h;
1737 %     elseif nargsout == 2
1738 %         varargout{1} = out_line_h;
1739 %         varargout{2} = out_axes_h;
1740 %     elseif nargsout == 3
1741 %         varargout{1} = out_line_h;
1742 %         varargout{2} = out_axes_h;
1743 %         varargout{3} = out_figure_h;
1744 %     else
1745 %         error('### Unknown number of outputs.')
1746 %     end
1747 %
1748 %
1749 % end % function varargout = iplot(varargin)
1750 %
1751 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1752 % %
1753 % % DESCRIPTION: GETDEFAULTLIST returns the default parameter list.
1754 % %
1755 % % CALL:         pl_default = getDefaultPL()
1756 % %
1757 % % HISTORY:      21-08-2007 Diepholz
1758 % %                Creation
1759 % %
1760 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1761 % function pl_default = getDefaultPL()
1762 %
1763 %     pl_default = plist();
1764 %     pl_default = append(pl_default, 'complex_view', 'mag/deg');
1765 %     pl_default = append(pl_default, 'enbw',         false);
1766 % end
1767 %
1768 %
1769 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1770 % %
1771 % % DESCRIPTION: WRAPLEGENDSTRING wrap a legend string into multiple lines.
1772 % %
1773 % % CALL:         so = wrapLegendString(s, N)
1774 % %
1775 % % HISTORY:      21-08-2007 M Hewitson
1776 % %                Creation
1777 % %
1778 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1779 % function so = wrapLegendString(s, N)
1780 %     % get cells
1781 %     sc = wrapstring(s, N);
1782 %
1783 %     so = sc{1};

```

```
1784 %  
1785 %   for j=2:length(sc)  
1786 %       so = [so sprintf('\n%s', sc{j})];  
1787 %   end  
1788 % end  
1789 %  
1790 %  
1791 %  
1792 %
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > isvalid.m

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 if a is a valid A0; 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')
```

VERSION: \$Id: isvalid.m,v 1.3 2007/12/19 20:32:16 hewitson Exp \$

HISTORY: 25-09-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```

0001 function varargout = isvalid(varargin)
0002
0003 % ISVALID tests if the given ao has all the correct fields of the correct type.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: ISVALID tests if the given ao has all the correct fields of the
0008 %               correct type.
0009 %
0010 % CALL:         isvalid(a);
0011 %
0012 %               Returns 1 if a is a valid AO; 0 otherwise.
0013 %
0014 %               The following call returns a parameter list object that
0015 %               contains the default parameter values:
0016 %               >> pl = isvalid(ao, 'Params')
0017 %
0018 %               The following call returns the cvs version number:
0019 %               >> ver = isvalid(ao, 'Version')
0020 %
0021 % VERSION:      $Id: isvalid.m,v 1.3 2007/12/19 20:32:16 hewitson Exp $
0022 %
0023 % HISTORY: 25-09-07 M Hewitson
0024 %           Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0027
0028 VERSION = '$Id: isvalid.m,v 1.3 2007/12/19 20:32:16 hewitson Exp $';
0029
0030 % 'Params' Call
0031 if nargin == 2
0032     if isa(varargin{1}, 'ao') && strcmp(varargin{2}, 'Params')
0033         varargout{1} = plist();
0034         return
0035     elseif isa(varargin{1}, 'ao') && strcmp(varargin{2}, 'Version')
0036         varargout{1} = VERSION;
0037         return
0038     end
0039 end
0040
0041 as = [];
0042 for j=1:nargin
0043     if isa(varargin{j}, 'ao')
0044         as = [as varargin{j}];
0045     end
0046 end
0047
0048 res = zeros(size(as));
0049 for j=1:numel(as)
0050     a = as(j);
0051     if ischar(a.name) && ...
0052         isa(a.hist, 'history') && ...

```

```
0053     isa(a.provenance, 'provenance') && ...
0054     ischar(a.description) && ...
0055     iscell(a.mfile) && ...
0056     ischar(a.mfilename) && ...
0057     ischar(a.mdlfile) && ...
0058     ischar(a.mdlfilename) && ...
0059     ischar(a.version) && ...
0060     isa(a.created, 'time')
0061
0062     res(j) = 1;
0063 else
0064     res(j) = 0;
0065 end
0066 end
0067
0068 varargout{1} = res;
0069
```

[Home](#) > [classes](#) > [@ao](#) > le.m

le

PURPOSE

LE overloads <= operator for analysis objects. Compare the y-axis values.

SYNOPSIS

function a = le(b,c)

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 1.5 2007/11/23 12:11:08 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

```
0001 function a = le(b,c)
0002 % LE overloads <= operator for analysis objects. Compare the y-axis values.
0003 %
```



```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LE overloads <= operator for analysis objects.
0007 %             Compare the y-axis values.
0008 %
0009 % CALL: a = b<=;
0010 %
0011 % POSSIBLE VALUES: b: Analysis object
0012 %                   c: Analysis object or a scalar.
0013 %
0014 % VERSION: $Id: le.m,v 1.5 2007/11/23 12:11:08 ingo Exp $
0015 %
0016 % HISTORY: 06-02-2007 M Hewitson
0017 %             Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: le.m,v 1.5 2007/11/23 12:11:08 ingo Exp $';
0022
0023 % Check if this is a call for parameters or for the cvs-version number
0024 if nargin == 2
0025     if isa(b, 'ao') && ischar(c)
0026         in = char(c);
0027         if strcmp(in, 'Params')
0028             a = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             a = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 %% Create parameter list to get the y-axis values. The value is not necessary.
0038 % It is only necessary that the value is set.
0039 pl = plist(param('ydata', 'x_xx_y'));
0040
0041 %% Get the data from analysis object
0042 [dummy, x] = get_xy_values(b.data, pl);
0043
0044 %% Get data from second input
0045 if isa(c, 'ao')
0046     [dummy, y] = get_xy_values(c.data, pl);
0047 else
0048     y = c*ones(size(x));
0049 end
0050
0051 %% Create Output
0052 a = x<=y;
0053
0054 % END

```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > len.m

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 1.8 2007/10/31 17:03:45 ingo Exp \$

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

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

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:

- **decimate** DECIMATE AOs containing time-series data.
- **submit_fastinsert** SUBMIT AO(s) to the MySQL server.

SUBFUNCTIONS

- **function** pl_default = getDefaultPL()

SOURCE CODE

```

0001 function data_len = len(varargin)
0002 % LEN overloads the length operator for Analysis objects. Length of the data
0003 % samples.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LEN overloads the length operator for Analysis objects.
0007 %               Number of data samples in the analysis object.
0008 %               LEN(ao) is the length of the elements of ao.data.
0009 %
0010 % CALL: l = len(ao_in);
0011 %
0012 % POSSIBLE VALUES: ao_in = [ao2 ao3]
0013 %                   ao_in = ao_vector
0014 %                   ao_in = ao_matrix
0015 %
0016 % VERSION: $Id: len.m,v 1.8 2007/10/31 17:03:45 ingo Exp $
0017 %
0018 % The following call returns a parameter list object that contains the
0019 % default parameter values:
0020 %
0021 % >> pl = len(ao, 'Params')
0022 %
0023 % HISTORY: 24-04-07 M Hewitson
0024 %           Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0027
0028 VERSION = '$Id: len.m,v 1.8 2007/10/31 17:03:45 ingo Exp $';
0029
0030 %% Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             data_len = getDefaultPL();
0036             return
0037         elseif strcmp(in, 'Version')
0038             data_len = VERSION;
0039             return

```

```
0040     end
0041 end
0042 end
0043
0044 %% store the input ao's in the vector: ao_set
0045 ao_set = [];
0046 for i=1:nargin
0047     a = varargin{i};
0048     if isa(a, 'ao')
0049         ao_set = [ao_set a];
0050     end
0051 end
0052
0053 data_len = zeros(size(ao_set));
0054
0055 %% go through analysis objects
0056 for j=1:numel(ao_set)
0057
0058     a = ao_set(j);
0059
0060     [x,y] = get_xy_values(a.data);
0061
0062     ld = length(y);
0063
0064     data_len(j) = ld;
0065 end
0066
0067 %% Get default params
0068 function pl_default = getDefaultPL()
0069
0070     pl_default = plist();
0071
0072 % END
```

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.

%%%

DESCRIPTION: LN overloads the log operator for Analysis objects.
Natural logarithm.
LN(ao) is the natural logarithm of the elements of ao.data.

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

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 = ln(ao, 'Params')

HISTORY: 23-05-2007 Diepholz
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- **char** CHAR overloads char() function for analysis objects.
- **log** LOG overloads the log operator for Analysis objects. Natural logarithm.

This function is called by:

SUBFUNCTIONS

- **function** pl_default = getDefaultPL()

SOURCE CODE

```

0001 function ao_out = ln (varargin)
0002 % LN overloads the log operator for Analysis objects. Natural logarithm.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LN overloads the log operator for Analysis objects.
0007 %               Natural logarithm.
0008 %               LN(ao) is the natural logarithm of the elements of ao.data.
0009 %
0010 % CALL: ao_out = ln(ao_in);
0011 %       ao_out = ln(ao_in, pl);
0012 %       ao_out = ln(ao1, pl1, ao_vector, ao_matrix, pl2);
0013 %
0014 % POSSIBLE VALUES: ao_in  = [ao2 ao3]
0015 %                   ao_in  = ao_vector
0016 %                   ao_in  = ao_matrix
0017 %
0018 % PARAMETER LIST:  <key>          <value>          <description>
0019 %                   tsdata    fsdata    xydata
0020 %                   'xdata'    't'       'f'       'x'       compute the xdata
0021 %                   'ydata'    'x'       'xx'      'y'       compute the ydata
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 % >> pl = ln(ao, 'Params')
0027 %
0028 % HISTORY: 23-05-2007 Diepholz
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032 %
0033 VERSION  = '$Id: ln.m,v 1.6 2007/10/31 17:03:45 ingo Exp $';
0034 ao_out   = [];
0035 pl       = [];
0036
0037 %% Check if this is a call for parameters

```

```

0038 if nargin == 2
0039     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0040         in = char(varargin{2});
0041         if strcmp(in, 'Params')
0042             ao_out = getDefaultPL();
0043             return
0044         elseif strcmp(in, 'Version')
0045             ao_out = VERSION;
0046             return
0047         end
0048     end
0049 end
0050
0051 %% store the input ao's in the vector: ao_set
0052 ao_set = [];
0053 for i=1:nargin
0054     a = varargin{i};
0055     if isa(a, 'ao')
0056         ao_set = [ao_set a];
0057     elseif isa(varargin{i}, 'plist')
0058         pl = [pl varargin{i}];
0059     end
0060 end
0061
0062 ao_out = log(ao_set, pl);
0063
0064 %% Get default params
0065 function pl_default = getDefaultPL()
0066
0067     pl_default = plist([param('xdata', '')
0068                         param('ydata', '')]);
0069
0070 % END

```


[Home](#) > [classes](#) > [@ao](#) > lt.m

lt

PURPOSE

LT overloads < operator for analysis objects. Compare the y-axis values.

SYNOPSIS

function a = lt(b,c)

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 1.5 2007/11/23 12:11:08 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

```
0001 function a = lt(b,c)
0002 % LT overloads < operator for analysis objects. Compare the y-axis values.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LT overloads < operator for analysis objects.
0007 %             Compare the y-axis values.
0008 %
0009 % CALL: a = b<c;
0010 %
0011 % POSSIBLE VALUES: b: Analysis object
0012 %                   c: Analysis object or a scalar.
0013 %
0014 % VERSION: $Id: lt.m,v 1.5 2007/11/23 12:11:08 ingo Exp $
0015 %
0016 % HISTORY: 06-02-2007 M Hewitson
0017 %             Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: lt.m,v 1.5 2007/11/23 12:11:08 ingo Exp $';
0022
0023 % Check if this is a call for parameters or for the cvs-version number
0024 if nargin == 2
0025     if isa(b, 'ao') && ischar(c)
0026         in = char(c);
0027         if strcmp(in, 'Params')
0028             a = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             a = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 %% Create parameter list to get the y-axis values. The value is not necessary.
0038 % It is only necessary that the value is set.
0039 pl = plist(param('ydata', 'x_xx_y'));
0040
0041 %% Get the data from analysis object
0042 [dummy, x] = get_xy_values(b.data, pl);
0043
0044 %% Get data from second input
0045 if isa(c, 'ao')
0046     [dummy, y] = get_xy_values(c.data, pl);
0047 else
0048     y = c*ones(size(x));
0049 end
0050
0051 %% Create Output
0052 a = x<y;
0053
0054
0055

```

0056 % END

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > md5.m

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 1.3 2007/10/26 16:09:20 ingo Exp \$

HISTORY: 15-09-2007 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xml](#) XML convert an analysis object to an DOM XML object.

This function is called by:

SOURCE CODE

```
0001 function h = md5(varargin)
0002 % MD5 computes an MD5 checksum from an analysis objects.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MD5 computes an MD5 checksum from an analysis objects.
0007 %
0008 % CALL:          h = md5(a)
0009 %
0010 % INPUTS:        a - input analysis object
0011 %
0012 % OUTPUTS:       h - md5 hash
0013 %
0014 % VERSION:       $Id: md5.m,v 1.3 2007/10/26 16:09:20 ingo Exp $
0015 %
0016 % HISTORY:       15-09-2007 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: md5.m,v 1.3 2007/10/26 16:09:20 ingo Exp $';
0022
0023 % 'Params' Call
0024 if nargin == 2
0025     if isa(varargin{1}, 'ao') && strcmp(varargin{2}, 'Params')
0026         h = plist();
0027         return
0028     elseif isa(varargin{1}, 'ao') && strcmp(varargin{2}, 'Version')
0029         h = VERSION;
0030         return
0031     end
0032 end
0033
0034 as = [];
0035 for j=1:nargin
0036     if isa(varargin{j}, 'ao')
0037         as = [as varargin{j}];
0038     end
0039 end
0040
0041 h = {};
0042
0043 for ii = 1:numel(as)
0044     x = xml(as(ii));
0045     h = [h cellstr(ltpda_hash(xmlwrite(x.docNode), 'MD5'))];
0046 end
0047
0048 h = reshape(h, size(as));
0049
0050 if numel(h) == 1
0051     h = cell2mat(h);
0052 end
0053

```


[Home](#) > [classes](#) > [@ao](#) > mux.m

mux

PURPOSE

MUX concatonate AOs into a vector.

SYNOPSIS

function b = mux(varargin)

DESCRIPTION

MUX concatonate AOs into a vector.

%%%

DESCRIPTION: MUX concatonate AOs into a vector.

CALL: ao_vec = mux(a1 ,a2);

VERSION: \$Id: mux.m,v 1.5 2007/12/11 17:27:09 ingo Exp \$

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

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

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

```

0001 function b = mux(varargin)
0002 % MUX concatonate AOs into a vector.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MUX concatonate AOs into a vector.
0007 %
0008 % CALL:          ao_vec = mux(a1 ,a2);
0009 %
0010 % VERSION:       $Id: mux.m,v 1.5 2007/12/11 17:27:09 ingo Exp $
0011 %
0012 % The following call returns a parameter list object that contains the
0013 % default parameter values:
0014 %
0015 % >> pl = mux(ao, 'Params')
0016 %
0017 % HISTORY: 27-03-07 M Hewitson
0018 %          Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: mux.m,v 1.5 2007/12/11 17:27:09 ingo Exp $';
0023
0024 % Check if this is a call for parameters
0025 if nargin == 2
0026     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0027         in = char(varargin{2});
0028         if strcmp(in, 'Params')
0029             b = getDefaultPL();
0030             return
0031         elseif strcmp(in, 'Version')
0032             b = VERSION;
0033             return
0034         end
0035     end
0036 end
0037
0038 b = [];
0039 for j=1:nargin
0040     if isa(varargin{j}, 'ao')
0041         b = [b varargin{j}];
0042     end
0043 end
0044
0045 %% Get default params
0046 function pl_default = getDefaultPL()
0047
0048     pl_default = plist();

```


Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > ne.m

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')

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 1.6 2007/11/23 12:11:08 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

```

0001 function result = ne(c1,c2, varargin)
0002
0003 % NE overloads the ~= operator for analysis objects.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: NE overloads the ~= operator for analysis objects.
0008 %
0009 % CALL:          result = ne(c1,c2)
0010 %               result = ne(c1,c2,  exc_list)
0011 %               result = ne(c1,c2, 'property1', 'property2')
0012 %               result = ne(c1,c2, 'class/property', 'class/property')
0013 %
0014 % EXAMPLES:      result = ne(c1,c2, 'name', 'created')
0015 %               result = ne(c1,c2, 'ao/name')
0016 %
0017 % INPUTS:        c1,c2      - input analysis objects
0018 %               exc_list - exception list
0019 %                   List of properties which are not checked.
0020 %
0021 % OUTPUTS:       If the two analysis objects are considered equal, result == 0,
0022 %               otherwise, result == 1.
0023 %
0024 % VERSION:       $Id: ne.m,v 1.6 2007/11/23 12:11:08 ingo Exp $
0025 %
0026 % HISTORY:       29-08-2007 M Hewitson
0027 %               Creation
0028 %
0029 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0030
0031 VERSION = '$Id: ne.m,v 1.6 2007/11/23 12:11:08 ingo Exp $';
0032
0033 % Check if this is a call for parameters or for the cvs-version number
0034 if nargin == 2
0035     if isa(c1, 'ao') && ischar(c2)
0036         in = char(c2);
0037         if strcmp(in, 'Params')
0038             result = plist();
0039             return
0040         elseif strcmp(in, 'Version')
0041             result = VERSION;
0042             return
0043         end
0044     end
0045 end

```

```
0046
0047 if eq(c1,c2, varargin{:})
0048     result = 0;
0049 else
0050     result = 1;
0051 end
0052
0053 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > plot.m

plot

PURPOSE

PLOT a simple plot of analysis objects.

SYNOPSIS

function varargout = **plot**(varargin)

DESCRIPTION

PLOT a simple plot of analysis objects.

%%%

DESCRIPTION: PLOT a simple plot of analysis objects.

CALL:

```

plot(ao)
plot(ao,LineStyle)
plot(axes_handle, ao)
plot(axes_handle, ao, LineSpec)
plot(..., 'PropertyName',PropertyValue,...)
[line_h]           = plot(...)
[line_h, axes_h]   = plot(...)
[line_h, axes_h, figure_h] = plot(...)
```

VERSION: \$Id: plot.m,v 1.30 2007/10/24 17:35:28 ingo Exp \$

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

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

HISTORY: 16-08-2007 M Diepholz
Creation

%%%

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.
- **plot_old** PLOT plots an array of analysis objects.
- **simple_plot** PLOT a simple plot of analysis objects.

SOURCE CODE

```

0001 function varargout = plot(varargin)
0002 % PLOT a simple plot of analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: PLOT a simple plot of analysis objects.
0007 %
0008 % CALL:          plot(ao)
0009 %                plot(ao,LineStyle)
0010 %                plot(axes_handle, ao)
0011 %                plot(axes_handle, ao, LineSpec)
0012 %                plot(...,'PropertyName',PropertyValue,...)
0013 %                [line_h] = plot(...)
0014 %                [line_h, axes_h] = plot(...)
0015 %                [line_h, axes_h, figure_h] = plot(...)
0016 %
0017 % VERSION:       $Id: plot.m,v 1.30 2007/10/24 17:35:28 ingo Exp $
0018 %
0019 % The following call returns a parameter list object that contains the
0020 % default parameter values:
0021 %
0022 % >> pl = plot(ao, 'Params')
0023 %
0024 % HISTORY:       16-08-2007 M Diepholz
0025 %                Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 VERSION = '$Id: plot.m,v 1.30 2007/10/24 17:35:28 ingo Exp $';
0030
0031 axes_handle = [];
0032 ao_s        = [];
0033 line_spec   = '';
0034 prop_val    = {};
0035 lines_h     = [];
0036
0037 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Check if this is a call for parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0038 if nargin == 2

```

```

0039     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0040         in = char(varargin{2});
0041         if strcmp(in, 'Params')
0042             varargout{1} = plist();
0043             return
0044         elseif strcmp(in, 'Version')
0045             varargout{1} = VERSION;
0046             return
0047         end
0048     end
0049 end
0050
0051 %%%%%%%%%% Check input values %%%%%%%%%%
0052 var_argin = varargin;
0053
0054 while ~isempty(var_argin)
0055     if isempty(var_argin{1})
0056         var_argin = var_argin(2:end);
0057     elseif ishandle(var_argin{1}) & isempty(axes_handle)
0058         axes_handle = var_argin{1};
0059         var_argin = var_argin(2:end);
0060     elseif isa(var_argin{1}, 'ao')
0061         var_argin{1} = reshape(var_argin{1}, 1, []);
0062         ao_s = [ao_s var_argin{1}];
0063         var_argin = var_argin(2:end);
0064     elseif ischar(var_argin{1}) && islinespec(var_argin{1}) && isempty
(line_spec)
0065         line_spec = var_argin{1};
0066         var_argin = var_argin(2:end);
0067     else
0068         if length(var_argin) == 1
0069             error('### There is only one property [%s] without value left', var_argin
{1});
0070         elseif length(var_argin) >= 2
0071             prop = var_argin{1};
0072             val = var_argin{2};
0073             prop_val = [prop_val, prop, val];
0074             var_argin = var_argin(3:end);
0075         end
0076     end
0077 end
0078
0079 %%%%%%%%%%
0080 % For all ao_s %
0081 %%%%%%%%%%
0082 for ii = 1:length(ao_s)
0083
0084     %%%%%%%%%% Set the line colors %%%%%%%%%%
0085     line_colors = getappdata(0, 'ltpda_default_plot_colors');
0086
0087     %%% Set only the color if the color is not set in the linespec
0088     [res, style] = islinespec(line_spec);
0089

```

```

0090     if isempty(style{3})
0091         prop_val{end+1} = 'Color';
0092         prop_val{end+1} = line_colors{ii};
0093     end
0094
0095     %%%%%%%%%%% Get the x-, y-values %%%%%%%%%%%
0096     [x,y] = get_xy_values(ao_s(ii).data);
0097
0098     if isreal(y)
0099         if isempty(x)
0100             x = 1:length(y);
0101         end
0102     else % iscomplex(y)
0103         if isempty(x)
0104             x = real(y);
0105             y = imag(y);
0106         end
0107     end
0108
0109     %%%%%%%%%%% Call the different plot command lines %%%%%%%%%%%
0110     if ~isempty(axes_handle) && ~isempty(line_spec)
0111         line_h = plot(axes_handle, x, y, line_spec);
0112     elseif ~isempty(axes_handle)
0113         line_h = plot(axes_handle, x, y);
0114     elseif ~isempty(line_spec)
0115         line_h = plot(x, y, line_spec);
0116     else
0117         line_h = plot(x, y);
0118     end
0119
0120     hold on;
0121
0122     line_h = reshape(line_h, 1, []);
0123     lines_h = [lines_h line_h];
0124
0125     %%%%%%%%%%% Set the different handles: axes- and figure- handle %%%%%%%%%%%
0126     axes_h = get(line_h(1), 'Parent');
0127     figure_h = get(axes_h, 'Parent');
0128
0129     %%%%%%%%%%% Set line series Properties OR axes Properties %%%%%%%%%%%
0130     while length(prop_val) >= 2
0131         prop = prop_val{1};
0132         val = prop_val{2};
0133         prop_val = prop_val(3:end);
0134         if isprop(line_h, prop)
0135             set(line_h, prop, val);
0136         else
0137             if isprop(axes_h, prop)
0138                 set(axes_h, prop, val);
0139             else
0140                 error('### Invalid line property OR axes property: '%s'', prop);

```



```

0141         end
0142     end
0143 end
0144
0145 end % for ii = 1:length(ao_s)
0146
0147
0148 %%%%%%%%%%% varargout{1} = line    handle %%%%%%%%%%%
0149 %%%%%%%%%%% varargout{2} = axes    handle %%%%%%%%%%%
0150 %%%%%%%%%%% varargout{3} = figure handle %%%%%%%%%%%
0151 if nargout == 0
0152 elseif nargout == 1
0153     varargout{1} = lines_h;
0154 elseif nargout == 2
0155     varargout{1} = lines_h;
0156     varargout{2} = axes_h;
0157 elseif nargout == 3
0158     varargout{1} = lines_h;
0159     varargout{2} = axes_h;
0160     varargout{3} = figure_h;
0161 else
0162     error('### Unknown number of outputs.')
0163 end
0164
0165 hold off;
0166
0167 end % function varargout = plot(varargin)
0168
0169
0170
0171
0172
0173
0174

```

[Home](#) > [classes](#) > [@ao](#) > plot_old.m

plot_old

PURPOSE

PLOT plots an array of analysis objects.

SYNOPSIS

function varargout = **plot**(as, varargin)

DESCRIPTION

PLOT plots an array of analysis objects.

%%%

DESCRIPTION: PLOT plots an array of analysis objects.

CALL: plot(a) - plots the analysis objects. One figure per data type.
 h = plot(a) - returns handles to figures
 [h,p] = plot(a) - returns handles to figures and plots

PARAMETERS: 'view' - 'mag/deg' 're/im' - used for complex series

VERSION: \$Id: plot_old.m,v 1.1 2007/10/08 09:47:08 ingo Exp \$

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

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

HISTORY: 01-02-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [abs](#) ABS overloads the Absolute value operator for Analysis objects.
- [char](#) CHAR overloads char() function for analysis objects.
- [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.
- **stairs** STAIRS overloads the stairs function for Analysis Objects.

This function is called by:

SUBFUNCTIONS

- function [hfig, handles] = ao_plot_fsdata(as, args)
- function [hfig, handles] = ao_plot_tsdata(as, args)
- function [hfig, handles] = ao_plot_cdata(as, args)
- function [hfig, handles] = ao_plot_xydata(as, args)
- function so = wrapLegendString(s, N)
- function pl_default = getDefaultPL()

SOURCE CODE

```

0001 function varargout = plot(as, varargin)
0002 % PLOT plots an array of analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: PLOT plots an array of analysis objects.
0007 %
0008 % CALL:          plot(a) - plots the analysis objects. One figure per data type.
0009 %              h = plot(a) - returns handles to figures
0010 %              [h,p] = plot(a) - returns handles to figures and plots
0011 %
0012 % PARAMETERS: 'view' - 'mag/deg' 're/im' - used for complex series
0013 %
0014 % VERSION:      $Id: plot_old.m,v 1.1 2007/10/08 09:47:08 ingo Exp $
0015 %
0016 % The following call returns a parameter list object that contains the
0017 % default parameter values:
0018 %
0019 % >> pl = plot(ao, 'Params')
0020 %
0021 % HISTORY: 01-02-07 M Hewitson
0022 %          Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025 %
0026 % 'xrange' - [x1 x2] set x-axis to this range (TODO)
0027 % 'yrange' - [y1 y2] set y-axis to this range (TODO)
0028 % 'xscale' - 'lin' or 'log' (TODO)
0029 % 'yscale' - 'lin' or 'log' (TODO)
0030
0031
0032 %% Check if this is a call for parameters
0033 if nargin == 2
0034     if isa(as, 'ao') && ischar(varargin{1})

```

```

0035     in = char(varargin{1});
0036     if strcmp(in, 'Params')
0037         varargout{1} = getDefaultPL();
0038         return
0039     end
0040 end
0041 end
0042
0043 %% check the ao's are all the same type or split them up into type groups
0044
0045 its = [];
0046 ifs = [];
0047 ic  = [];
0048 ixy = [];
0049
0050 % outputs
0051 hfig    = [];
0052 handles = [];
0053
0054 si = size(as);
0055 na = si(1);
0056 nb = si(2);
0057
0058 for j=1:nb
0059     for i=1:na
0060
0061         % what type of analysis object
0062         a = as(i,j);
0063         if iscell(a)
0064             a = a{1};
0065         end
0066         d = a.data;
0067         dinfo = whos('d');
0068         switch dinfo.class
0069             case 'tsdata'
0070                 its = [its sub2ind(si, i, j)];
0071             case 'fsdata'
0072                 ifs = [ifs sub2ind(si, i, j)];
0073             case 'cdata'
0074                 ic = [ic sub2ind(si, i, j)];
0075             case 'xydata'
0076                 ixy = [ixy sub2ind(si, i, j)];
0077             otherwise
0078                 error('### unknown data type for analysis objects.')
0079         end
0080     end
0081 end
0082
0083 % Plot all time-series (tsdata) objects
0084 [tfig, thandles] = ao_plot_tsdata(as(its), varargin);
0085 hfig    = [hfig tfig];
0086 handles = [handles thandles];

```

```

0087
0088 % Plot all freq-series (fsdata) objects
0089 [tfig, thandles] = ao_plot_fsdata(as(ifs), varargin);
0090 hfig    = [hfig tfig];
0091 handles = [handles thandles];
0092
0093 % Plot all cdata objects
0094 [tfig, thandles] = ao_plot_cdata(as(ic), varargin);
0095 hfig    = [hfig tfig];
0096 handles = [handles thandles];
0097
0098 % Plot all xydata objects
0099 [tfig, thandles] = ao_plot_xydata(as(ixy), varargin);
0100 hfig    = [hfig tfig];
0101 handles = [handles thandles];
0102
0103 % set the title of current figure
0104 global sltpda_loop_plotTitle;
0105 if ~isempty(sltpda_loop_plotTitle)
0106     set(gcf, 'Name', sltpda_loop_plotTitle);
0107 end
0108
0109 if nargout == 1
0110     varargout{1} = handles;
0111 end
0112
0113 %-----
0114 %
0115 %
0116 function [hfig, handles] = ao_plot_fsdata(as, args)
0117 % AO_PLOT_FSDATA plots freq-series type analysis objects.
0118 %
0119
0120 % outputs
0121 hfig    = [];
0122 handles = [];
0123
0124 % Default parameters
0125 view = 'mag/deg';
0126 lenLegend = getappdata(0, 'wraplegendstringat');
0127
0128 % get additional input parameters
0129 while length(args) >= 2
0130     prop = args{1};
0131     val  = args{2};
0132     args = args(3:end);
0133     switch prop
0134     case 'view'
0135         view = val;
0136     otherwise
0137         warning('!!! unknown parameter %s - ignoring.', prop);
0138     end

```

```

0139
0140 end
0141
0142 % Choose axes type
0143 na = length(as);
0144 haveComplex = 0;
0145 logy      = 0;
0146 for i=1:na
0147     a = as(i);
0148     d = a.data;
0149     if max(abs(d.xx))/min(abs(d.xx)) > 10
0150         logy = 1;
0151     end
0152     if ~isreal(d.xx)
0153         haveComplex = 1;
0154     end
0155 end
0156
0157 if na > 0
0158
0159     disp('--- plotting frequency-series data A0');
0160     colors = getappdata(0,'ltpda_default_plot_colors');
0161
0162 % hfig = figure;
0163 legendStr = [];
0164
0165 for i=1:na
0166     a = as(i);
0167
0168     % get data out
0169     d = a.data;
0170     f = d.f;
0171     xx = d.xx;
0172     col = colors{mod(i-1,length(colors))+1};
0173
0174     % plot real frequency series
0175     if ~haveComplex
0176         % do we need a subplot for ENBW?
0177         if length(d.enbw) == length(d.xx) && length(d.xx) > 1
0178             subplot(3,1,1:2)
0179             legendStr = [legendStr cellstr(wrapLegendString(sprintf('%s',
ltpda_label(a.name)), lenLegend))];
0180         else
0181             legendStr = [legendStr...
0182                         cellstr([...
0183                                 wrapLegendString(sprintf('%s', ltpda_label
(a.name)), lenLegend)...
0184                                 sprintf('\nenbw=%g', d.enbw)...
0185                                 ])...
0186                         ];
0187         end
0188

```

```

0189     if ~logy
0190         hp = semilogx(f, xx);
0191     else
0192         hp = loglog(f,xx);
0193     end
0194     set(hp, 'Color', col);
0195     hold on;
0196     title('Plot of frequency-series A0s');
0197     grid on;
0198     legend(legendStr)
0199     ylabel(sprintf('Amplitude [%s]', d.yunits));
0200     handles = [handles hp];
0201     set(gca, 'XScale', 'log');
0202     set(gca, 'YScale', 'log');
0203
0204
0205     % do we need a subplot for ENBW?
0206     if length(d.enbw) == length(d.xx) && length(d.xx) > 1
0207         disp(' + making ENBW plot');
0208         subplot(3,1,3)
0209         hp = stairs(f, d.enbw);
0210         hold on;
0211         set(hp, 'Color', col);
0212         grid on;
0213         ylabel('ENBW [Hz]');
0214         handles = [handles hp];
0215         set(gca, 'XScale', 'log');
0216         set(gca, 'YScale', 'lin');
0217     end
0218
0219     xlabel(sprintf('Frequency [%s]', d.xunits));
0220
0221 else
0222     % plot complex frequency series
0223     switch view
0224     case 'mag/deg'
0225         subplot(3,1,1:2)
0226         if ~logy
0227             hp = semilogx(f, abs(xx));
0228         else
0229             hp = loglog(f,abs(xx));
0230         end
0231
0232     %         hp = loglog(f, abs(xx));
0233     set(hp, 'Color', col);
0234     hold on;
0235     legendStr = [legendStr cellstr(wrapLegendString(sprintf('%s',
ltpda_label(a.name)), lenLegend))];
0236     title('Plot of complex frequency series A0s');
0237     grid on;
0238     handles = [handles hp];
0239     ylabel('Magnitude');

```

```

0240         legend(legendStr)
0241 %         set(gca, 'XScale', 'log');
0242 %         set(gca, 'YScale', 'log');
0243
0244         subplot(3,1,3)
0245         hp = semilogx(f, ltpda_phase(xx));
0246         set(hp, 'Color', col);
0247         hold on;
0248         grid on;
0249         xlabel(sprintf('Frequency [%s]', d.xunits));
0250         ylabel('Phase [deg]');
0251         handles = [handles hp];
0252 %         set(gca, 'XScale', 'log');
0253 %         set(gca, 'YScale', 'lin');
0254
0255     case 're/im'
0256         subplot(2,1,1)
0257         hp = semilogx(f, real(xx));
0258         set(hp, 'Color', col);
0259         hold on;
0260         title('Plot of complex frequency series AOs');
0261         legendStr = [legendStr cellstr(wrapLegendString(sprintf('%s',
ltpda_label(a.name)), lenLegend))];
0262         grid on;
0263         handles = [handles hp];
0264         ylabel('Real');
0265         legend(legendStr)
0266         set(gca, 'XScale', 'log');
0267         set(gca, 'YScale', 'lin');
0268
0269         subplot(2,1,2)
0270         hp = semilogx(f, imag(xx));
0271         set(hp, 'Color', col);
0272         hold on;
0273         grid on;
0274         xlabel(sprintf('Frequency [%s]', d.xunits));
0275         ylabel('Imag');
0276         handles = [handles hp];
0277         set(gca, 'XScale', 'log');
0278         set(gca, 'YScale', 'lin');
0279
0280     otherwise
0281         error('### unknown view for plotting complex fsdata.');
```

```

0282 end
0283 ltpda_allxaxis(min(f), max(f));
0284 % subplot(3,1,1:2)
0285 % legend(legendStr)
0286
0287 end
0288
0289 end % end loop over AOs

```



```

0290 end
0291
0292
0293 %-----
0294 %
0295 %
0296 function [hfig, handles] = ao_plot_tsdata(as, args)
0297 % AO_PLOT_TSDATA plots time-series type analysis objects.
0298 %
0299
0300
0301 % outputs
0302 hfig = [];
0303 handles = [];
0304 lenLegend = getappdata(0, 'wraplegendstringat');
0305
0306 na = length(as);
0307 if na > 0
0308
0309     % Compute absolute time offset
0310     Toff = 1e20;
0311     for j=1:na
0312         t0 = as(j).data.t0.utc_epoch_milli/1000;
0313         if t0 < Toff
0314             Toff = t0;
0315         end
0316     end
0317
0318     disp('--- plotting time-series data A0');
0319     colors = getappdata(0, 'ltpda_default_plot_colors');
0320
0321     handles = [];
0322
0323 %     hfig = figure;
0324     legendStr = [];
0325     for i=1:na
0326         a = as(i);
0327         if iscell(a)
0328             a = a{1};
0329         end
0330         % get data out
0331         d = a.data;
0332         t0 = a.data.t0.utc_epoch_milli/1000;
0333         t = d.t + t0 - Toff;
0334         x = d.x;
0335
0336         col = colors{mod(i-1, length(colors))+1};
0337         hp = plot(t,x);
0338         set(hp, 'Color', col);
0339         hold on;
0340         legendStr = [legendStr...
0341                     cellstr([wrapLegendString(sprintf('%s', ltpda_label(a.name)),

```

```

lenLegend)...
0342                                     sprintf('\n%s\n%6.2f secs @ %gHz', char(d.
t0), d.nsecs, d.fs))]];
0343
0344     title('Plot of time-series AOs');
0345     grid on;
0346     legend(legendStr)
0347     xlabel(sprintf('Time from %s [%s]', char(time(Toff*1000)), d.xunits));
0348     ylabel(sprintf('Amplitude [%s]', d.yunits));
0349     set(gca, 'XScale', 'lin');
0350     set(gca, 'YScale', 'lin');
0351     handles = [handles hp];
0352 end
0353 end
0354
0355 %-----
0356 %
0357 %
0358 function [hfig, handles] = ao_plot_cdata(as, args)
0359 % AO_PLOT_CDATA plots cdata type analysis objects.
0360 %
0361
0362
0363 % outputs
0364 hfig    = [];
0365 handles = [];
0366 lenLegend = getappdata(0, 'wraplegendstringat');
0367
0368 na = length(as);
0369
0370 if na > 0
0371     disp('--- plotting cdata AO');
0372     colors = getappdata(0, 'ltpda_default_plot_colors');
0373
0374     handles = [];
0375
0376 %     hfig = figure;
0377     legendStr = [];
0378     for i=1:na
0379         a = as(i);
0380         if iscell(a)
0381             a = a{1};
0382         end
0383
0384         % get data out
0385         d = a.data;
0386         if isreal(d.vals)
0387             y = d.vals;
0388             x = 1:length(y);
0389         else
0390             x = real(d.vals);
0391             y = imag(d.vals);

```

```

0392     end
0393
0394     col = colors{mod(i-1,length(colors))+1};
0395     hp = plot(x,y,'o');
0396     set(hp, 'Color', col);
0397     hold on;
0398     legendStr = [legendStr cellstr(wrapLegendString(sprintf('%s', ltpda_label(a.
name))), lenLegend))];
0399
0400     title('Plot of cdata AOs');
0401     grid on;
0402     legend(legendStr)
0403     if isreal(d.vals)
0404         xlabel('Sample');
0405         ylabel('Value');
0406     else
0407         xlabel('Real');
0408         ylabel('Imag');
0409     end
0410     set(gca, 'XScale', 'lin');
0411     set(gca, 'YScale', 'lin');
0412 end
0413 end
0414
0415 %-----
0416 %
0417 %
0418 function [hfig, handles] = ao_plot_xydata(as, args)
0419 % AO_PLOT_XYDATA plots xydata type analysis objects.
0420 %
0421
0422 % outputs
0423 hfig    = [];
0424 handles = [];
0425 lenLegend = getappdata(0, 'wraplegendstringat');
0426
0427 na = length(as);
0428 if na > 0
0429     disp('--- plotting X-Y data AO');
0430     colors = getappdata(0, 'ltpda_default_plot_colors');
0431
0432     handles = [];
0433
0434 %     hfig = figure;
0435     legendStr = [];
0436     for i=1:na
0437         a = as(i);
0438         if iscell(a)
0439             a = a{1};
0440         end
0441         % get data out
0442         d = a.data;

```

```

0443     x = d.x;
0444     y = d.y;
0445
0446     col = colors{mod(i-1,length(colors))+1};
0447     hp = plot(x,y);
0448     set(hp, 'Color', col);
0449     hold on;
0450     legendStr = [legendStr cellstr(wrapLegendString(sprintf('%s', ltpda_label(a.
name))), lenLegend))];
0451
0452     title('Plot of xydata AOs');
0453     grid on;
0454     legend(legendStr)
0455     xlabel(sprintf('X [%s]', d.xunits));
0456     ylabel(sprintf('Y [%s]', d.yunits));
0457     set(gca, 'XScale', 'lin');
0458     set(gca, 'YScale', 'lin');
0459 end
0460 end
0461
0462 %-----
0463 % wrap a legend string into multiple lines
0464 %
0465 function so = wrapLegendString(s, N)
0466
0467 % get cells
0468 sc = wrapstring(s, N);
0469
0470 so = sc{1};
0471
0472 for j=2:length(sc)
0473     so = [so sprintf('\n%s', sc{j})];
0474 end
0475
0476 % so = cellstr(so);
0477
0478 %-----
0479 %% Get default params
0480 function pl_default = getDefaultPL()
0481
0482 disp('* creating default plist...');
0483     pl_default = plist();
0484 disp('* done. ');
0485
0486 % END

```

[Home](#) > [classes](#) > [@ao](#) > save.m

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 1.14 2008/01/07 20:41:19 hewitson Exp \$

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

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

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

```

0001 function varargout = save(varargin)
0002 % SAVE overloads save operator for analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SAVE overloads save operator for analysis objects.
0007 %
0008 % CALL:          Save an analysis object as a .mat file.
0009 %                >> save(a, 'blah.mat')
0010 %                Save an analysis object as an XML file.
0011 %                >> save(a, 'blah.xml')
0012 %
0013 % Parameters: 'filename' - name of the file
0014 %
0015 % VERSION:       $Id: save.m,v 1.14 2008/01/07 20:41:19 hewitson Exp $
0016 %
0017 % The following call returns a parameter list object that contains the
0018 % default parameter values:
0019 %
0020 % >> pl = save(ao, 'Params')
0021 %
0022 % HISTORY: 14-02-07 M Hewitson
0023 %          Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026 %
0027 %
0028 ALGONAME = mfilename;
0029 VERSION  = '$Id: save.m,v 1.14 2008/01/07 20:41:19 hewitson Exp $';
0030 %
0031 % Check if this is a call for parameters
0032 if nargin == 2
0033     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0034         in = char(varargin{2});
0035         if strcmp(in, 'Params')
0036             varargout{1} = getDefaultPL();
0037             return
0038         elseif strcmp(in, 'Version')
0039             varargout{1} = VERSION;
0040             return
0041         end
0042     end

```

```

0043 end
0044
0045 if nargin ~= 2
0046     help mfilename;
0047     error('### incorrect inputs');
0048 end
0049
0050 % get object
0051 obj = varargin{1};
0052 if isa(varargin{2}, 'plist')
0053     pl = varargin{2};
0054 elseif ischar(varargin{2})
0055     pl = plist(param('filename', varargin{2}));
0056 else
0057     help mfilename;
0058     error('### incorrect inputs');
0059 end
0060
0061 % Save object
0062 ltpda_saveobj(obj, pl);
0063
0064
0065 % % Check if this is a call for parameters
0066 % if nargin == 2
0067 %     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0068 %         in = char(varargin{2});
0069 %         if strcmp(in, 'Params')
0070 %             varargout{1} = getDefaultPL();
0071 %             return
0072 %         end
0073 %     end
0074 % end
0075 %
0076 %
0077 % ALGONAME = mfilename;
0078 % VERSION = '$Id: save.m,v 1.14 2008/01/07 20:41:19 hewitson Exp $';
0079 %
0080 % if nargin ~= 2
0081 %     help mfilename;
0082 %     error('### incorrect inputs');
0083 % end
0084 %
0085 % % analysis object
0086 % a = varargin{1};
0087 % if isa(varargin{2}, 'plist')
0088 %     filename = find(varargin{2}, 'filename');
0089 %     plo = varargin{2};
0090 % else
0091 %     % filename
0092 %     filename = varargin{2};
0093 %     plo = plist(param('filename', filename));
0094 % end

```

```
0095 %
0096 % % Inspect filename
0097 % [path,name,ext,vers] = fileparts(filename);
0098 %
0099 % switch ext
0100 %     case '.mat'
0101 %
0102 %         save(filename, 'a');
0103 %
0104 %     case '.xml'
0105 %
0106 % %         hist = a.hist;
0107 % %         h = history(ALGONAME, VERSION, plo, hist);
0108 % %
0109 % %         a = set(a, 'hist', h);
0110 % %         a = set(a, 'name', sprintf('save(%s,%s)', a.name, filename));
0111 % %
0112 % %         % convert AO to xml
0113 % %         x = xml(a);
0114 % %         % Save the XML document.
0115 % %         xmlwrite(filename,x.docNode);
0116 %
0117 %     otherwise
0118 %         error('### unknown file extension.');
```

```
0119 % end
0120 %
0121 % % no outputs
0122 % if nargout == 1
0123 %     varargout{1} = a;
0124 % else
0125 %     for j=1:nargout
0126 %         varargout{j} = [];
0127 %     end
0128 % end
0129
0130 % Get default params
0131 function plo = getDefaultPL()
0132
0133 plo = plist();
0134 plo = append(plo, param('filename', ''));
0135
0136 % END
```


[Home](#) > [classes](#) > [@ao](#) > simple_plot.m

simple_plot

PURPOSE

PLOT a simple plot of analysis objects.

SYNOPSIS

function varargout = **plot**(varargin)

DESCRIPTION

PLOT a simple plot of analysis objects.

%%%

DESCRIPTION: PLOT a simple plot of analysis objects.

CALL:

```

plot(ao)
plot(ao,LineStyle)
plot(axes_handle, ao)
plot(axes_handle, ao, LineSpec)
plot(..., 'PropertyName',PropertyValue,...)
[line_h] = plot(...)
[line_h, axes_h] = plot(...)
[line_h, axes_h, figure_h] = plot(...)

```

VERSION: \$Id: simple_plot.m,v 1.9 2007/10/24 17:35:28 ingo Exp \$

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

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

HISTORY: 16-08-2007 M Diepholz
Creation

%%%

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:

SOURCE CODE

```

0001 function varargout = plot(varargin)
0002 % PLOT a simple plot of analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: PLOT a simple plot of analysis objects.
0007 %
0008 % CALL:          plot(ao)
0009 %                plot(ao,LineStyle)
0010 %                plot(axes_handle, ao)
0011 %                plot(axes_handle, ao, LineSpec)
0012 %                plot(..., 'PropertyName', PropertyValue, ...)
0013 %                [line_h]                = plot(...)
0014 %                [line_h, axes_h]         = plot(...)
0015 %                [line_h, axes_h, figure_h] = plot(...)
0016 %
0017 % VERSION:       $Id: simple_plot.m,v 1.9 2007/10/24 17:35:28 ingo Exp $
0018 %
0019 % The following call returns a parameter list object that contains the
0020 % default parameter values:
0021 %
0022 % >> pl = plot(ao, 'Params')
0023 %
0024 % HISTORY:       16-08-2007 M Diepholz
0025 %                Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 axes_handle = [];
0030 ao_s       = [];
0031 line_spec  = '';
0032 prop_val   = {};
0033 lines_h    = [];
0034
0035 %%%%%%%%%%%%% Check if this is a call for parameters %%%%%%%%%%%%%
0036 if nargin == 2
0037     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0038         in = char(varargin{1});
0039         if strcmp(in, 'Params')
0040             varargout{1} = plist();
0041             return
0042         end
0043     end

```

```

0044 end
0045
0046 %%%%%%%%%% Check input values %%%%%%%%%%
0047 for ii = 1:nargin
0048     if ishandle(varargin{ii}) & isempty(axes_handle)
0049         axes_handle = varargin{ii};
0050     elseif isa(varargin{ii}, 'ao')
0051         varargin{ii} = reshape(varargin{ii}, 1, []);
0052         ao_s = [ao_s varargin{ii}];
0053     elseif ischar(varargin{ii}) && islinespec(varargin{ii}) && isempty
(line_spec)
0054         line_spec = varargin{ii};
0055     else
0056         prop_val = [prop_val, varargin{ii}];
0057     end
0058 end
0059
0060 %%%%%%%%%% For all ao_s %%%%%%%%%%
0061 %
0062 %%%%%%%%%%
0063 for ii = 1:length(ao_s)
0064
0065     %%%%%%%%%% Set the line colors %%%%%%%%%%
0066     line_colors = getappdata(0, 'ltpda_default_plot_colors');
0067
0068     %% Set only the color if the color is not set in the linespec
0069     [res, style] = islinespec(line_spec);
0070
0071     if isempty(style{3})
0072         prop_val{end+1} = 'Color';
0073         prop_val{end+1} = line_colors{ii};
0074     end
0075
0076     %%%%%%%%%% Get the x-, y-values %%%%%%%%%%
0077     [x,y] = get_xy_values(ao_s(ii).data);
0078
0079     if isreal(y)
0080         if isempty(x)
0081             x = 1:length(y);
0082         end
0083     else % iscomplex(y)
0084         if isempty(x)
0085             x = real(y);
0086             y = imag(y);
0087         end
0088     end
0089
0090     %%%%%%%%%% Call the different plot command lines %%%%%%%%%%
0091     if ~isempty(axes_handle) && ~isempty(line_spec)
0092         line_h = plot(axes_handle, x, y, line_spec);
0093     elseif ~isempty(axes_handle)
0094         line_h = plot(axes_handle, x, y);

```

```

0095     elseif ~isempty(line_spec)
0096         line_h = plot(x, y, line_spec);
0097     else
0098         line_h = plot(x, y);
0099     end
0100
0101     hold on;
0102
0103     line_h = reshape(line_h, 1, []);
0104     lines_h = [lines_h line_h];
0105
0106     %%%%%%%%%%% Set the different handles: axes- and figure- handle %%%%%%%%%%%
0107     axes_h = get(line_h(1), 'Parent');
0108     figure_h = get(axes_h, 'Parent');
0109
0110     %%%%%%%%%%% Set line series Properties OR axes Properties %%%%%%%%%%%
0111     while length(prop_val) >= 2
0112         prop = prop_val{1};
0113         val = prop_val{2};
0114         prop_val = prop_val(3:end);
0115         if isprop(line_h, prop)
0116             set(line_h, prop, val);
0117         else
0118             if isprop(axes_h, prop)
0119                 set(axes_h, prop, val);
0120             else
0121                 error('### Invalid line property OR axes property: '%s'', prop);
0122             end
0123         end
0124     end
0125
0126 end % for ii = 1:length(ao_s)
0127
0128
0129 %%%%%%%%%%% varargout{1} = line handle %%%%%%%%%%%
0130 %%%%%%%%%%% varargout{2} = axes handle %%%%%%%%%%%
0131 %%%%%%%%%%% varargout{3} = figure handle %%%%%%%%%%%
0132 if nargout == 0
0133 elseif nargout == 1
0134     varargout{1} = lines_h;
0135 elseif nargout == 2
0136     varargout{1} = lines_h;
0137     varargout{2} = axes_h;
0138 elseif nargout == 3
0139     varargout{1} = lines_h;
0140     varargout{2} = axes_h;
0141     varargout{3} = figure_h;
0142 else
0143     error('### Unknown number of outputs.')
0144 end
0145

```

```
0146     hold off;  
0147  
0148 end % function varargout = plot(varargin)  
0149  
0150  
0151  
0152  
0153  
0154  
0155
```

[Home](#) > [classes](#) > [@ao](#) > stairs.m

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)

OUTPUTS: h - plot handles

PARAMETERS: none

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

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

VERSION: \$Id: stairs.m,v 1.6 2007/11/06 16:31:49 ingo Exp \$

HISTORY: 02-03-07 M Hewitson
Creation.

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR overloads char() function for analysis objects.
- [set](#) SET set an analysis object property.

- **stairs** STAIRS overloads the stairs function for Analysis Objects.

This function is called by:

- **plot_old** PLOT plots an array of analysis objects.
- **stairs** STAIRS overloads the stairs function for Analysis Objects.

SUBFUNCTIONS

- **function plo = getDefaultPL()**

SOURCE CODE

```

0001 function varargout = stairs(varargin)
0002 % STAIRS overloads the stairs function for Analysis Objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: STAIRS overloads the stairs function for Analysis Objects.
0007 %
0008 % CALL:          stairs(a)
0009 %              h = stairs(a)
0010 %
0011 % INPUTS:        a - analysis object(s)
0012 %
0013 % OUTPUTS:       h - plot handles
0014 %
0015 % PARAMETERS:    none
0016 %
0017 %              The following call returns a parameter list object that
0018 %              contains the default parameter values:
0019 %
0020 %              >> pl = stairs(ao, 'Params')
0021 %
0022 % VERSION:       $Id: stairs.m,v 1.6 2007/11/06 16:31:49 ingo Exp $
0023 %
0024 % HISTORY: 02-03-07 M Hewitson
0025 %          Creation.
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 VERSION = '$Id: stairs.m,v 1.6 2007/11/06 16:31:49 ingo Exp $';
0030
0031 % Check if this is a call for parameters
0032 if nargin == 2
0033     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0034         in = char(varargin{2});
0035         if strcmp(in, 'Params')
0036             varargout{1} = getDefaultPL();
0037         return

```

```

0038     elseif strcmp(in, 'Version')
0039         varargout{1} = VERSION;
0040         return
0041     end
0042 end
0043 end
0044
0045 %----- Standard history variable
0046
0047 %----- capture input variable names, AOs, and plists
0048 invars = {};
0049 as      = [];
0050 ps      = [];
0051 Nbins   = -1;
0052 Xbins   = [];
0053 for j=1:nargin
0054     if isa(varargin{j}, 'ao')
0055         invars = [invars cellstr(inputname(j))];
0056     end
0057     if isa(varargin{j}, 'ao')
0058         as = [as varargin{j}];
0059     end
0060     if isa(varargin{j}, 'plist')
0061         ps = [ps varargin{j}];
0062     end
0063     if isnumeric(varargin{j})
0064         if length(varargin{j}) > 1
0065             Xbins = varargin{j};
0066         else
0067             Nbins = varargin{j};
0068         end
0069     end
0070 end
0071
0072 % produce one plist
0073 if isa(ps, 'plist')
0074     pl = combine(ps, getDefaultPL());
0075 else
0076     pl = getDefaultPL();
0077 end
0078
0079 % If we have X from command input, override plist
0080 if ~isempty(Xbins)
0081     pl = pset(pl, 'X', Xbins);
0082 end
0083
0084 % If we have N from command input, override plist
0085 if Nbins>0
0086     pl = pset(pl, 'N', Nbins);
0087 end
0088
0089 % Get parameters

```



```

0090 % N = find(pl, 'N');
0091 % X = find(pl, 'X');
0092
0093 colors = getappdata(0, 'ltpda_default_plot_colors');
0094
0095 %----- Loop over input AOs
0096
0097 % Initialise output
0098 handles = [];
0099 % start looping
0100 for j=1:numel(as)
0101     a = as(j);
0102     legendStr = [];
0103
0104     % get data
0105     [x,y] = get_xy_values(a.data);
0106     if isempty(x)
0107         x = 1:length(y);
0108     end
0109
0110     % make stair plot
0111     col = colors{mod(j-1,length(colors))+1};
0112     h = stairs(x,y);
0113     legendStr = [legendStr cellstr(sprintf('%s', ltpda_label(a.name)))];
0114     set(h, 'Color', col);
0115     title('Stair plot');
0116     grid on;
0117     legend(legendStr)
0118     ylabel(sprintf('Y [%s]', a.data.yunits));
0119     xlabel(sprintf('X [%s]', a.data.xunits));
0120     hold on;
0121
0122     % store handles for output
0123     handles = [handles h];
0124
0125 end
0126
0127 %----- Set outputs
0128 if nargin == 1
0129     varargout{1} = handles;
0130 end
0131 %-----
0132 % Get default params
0133 function plo = getDefaultPL()
0134
0135 plo = plist();
0136 % plo = append(plo, 'N', '');
0137 % plo = append(plo, 'X', '');
0138
0139
0140 % END

```


[Home](#) > [classes](#) > [@ao](#) > string.m

string

PURPOSE

STRING writes a command string that can be used to recreate the input Analysis object(s).

SYNOPSIS

function cmd = string(as)

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 1.4 2007/08/14 09:21:13 ingo 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

```
0001 function cmd = string(as)
```

```

0002 % STRING writes a command string that can be used to recreate the input
Analysis object(s).
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0007 %             input Analysis object(s).
0008 %
0009 % CALL:       cmd = string(a1)
0010 %
0011 % VERSION:    $Id: string.m,v 1.4 2007/08/14 09:21:13 ingo Exp $
0012 %
0013 % HISTORY: 29-03-07 M Hewitson
0014 %             Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 cmd = '[';
0019
0020 for j=1:length(as)
0021     a = as(j);
0022
0023     hist = get(a, 'hist');
0024     pl    = get(hist, 'plist');
0025
0026     if isempty(pl)
0027         error('### this AO was not created with a plist. Can''t convert to
string. ');
0028     end
0029
0030     plstr = string(pl);
0031
0032     cmd = [cmd 'ao(' ...
0033           plstr ...
0034           ') '];
0035 end
0036
0037 cmd = [cmd '']];
0038
0039 if strcmp(cmd, '[]')
0040     cmd = '';
0041 end
0042
0043 % END

```

[Home](#) > [classes](#) > [@ao](#) > submit_fastinsert.m

submit_fastinsert

PURPOSE

SUBMIT AO(s) to the MySQL server.

SYNOPSIS

function [AOids, Cid] = submit_fastinsert(varargin)

DESCRIPTION

SUBMIT AO(s) to the MySQL server.

%%%

DESCRIPTION: SUBMIT AO(s) to the MySQL server.
The input AOs will be submitted to the MySQL server configured in
ltpda_startup.m. If multiple AOs are submitted together,
a corresponding AO Collection entry will be made.

CALL: submit(a1, 'some comment')
 submit(a1,a2, 'some comment')
 submit([a1 a2], a3, 'some comment')

VERSION: \$Id: submit.m,v 1.12 2007/07/31 06:14:34 hewitson Exp \$

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

```
>> pl = submit('Params')
```

HISTORY: 09-05-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR overloads char() function for analysis objects.
- [get](#) GET get ao properties.
- [len](#) LEN overloads the length operator for Analysis objects. Length of the data samples.
- [xml](#) XML convert an analysis object to an DOM XML object.

This function is called by:

SUBFUNCTIONS

- function s = csv(x)
- function plo = getDefaultPL()
- function message = minsert(conn, table, varargin)

SOURCE CODE

```

0001 function [A0ids, Cid] = submit_fastinsert(varargin)
0002 % SUBMIT A0(s) to the MySQL server.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBMIT A0(s) to the MySQL server.
0007 %             The input A0s will be submitted to the MySQL server configured in
0008 %             ltpda_startup.m. If multiple A0s are submitted together,
0009 %             a corresponding A0 Collection entry will be made.
0010 %
0011 % CALL:       submit(a1, 'some comment')
0012 %             submit(a1,a2, 'some comment')
0013 %             submit([a1 a2], a3, 'some comment')
0014 %
0015 % VERSION:    $Id: submit.m,v 1.12 2007/07/31 06:14:34 hewitson Exp $
0016 %
0017 % The following call returns a parameter list object that contains the
0018 % default parameter values:
0019 %
0020 % >> pl = submit('Params')
0021 %
0022 % HISTORY: 09-05-07 M Hewitson
0023 %          Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026 %
0027 % Check if this is a call for parameters
0028 if nargin == 2
0029     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0030         in = char(varargin{2});
0031         if strcmp(in, 'Params')
0032             A0ids = getDefaultPL();
0033             return
0034         end
0035     end
0036 end
0037
0038 %-----
0039 % collect all A0s from input
0040 as      = [];

```

```

0041 comment = [];
0042 for j=1:nargin
0043     a = varargin{j};
0044     if isa(a, 'ao')
0045         as = [as a];
0046     end
0047     if ischar(a)
0048         comment = varargin{j};
0049     end
0050 end
0051 na = length(as);
0052 if na < 1
0053     error('### Incorrect inputs: please supply at least one A0 to submit.');
```

```

0054 end
0055 if isempty(comment) || length(comment)<10
0056     error('### Incorrect inputs: please supply a comment of at least 10
characters in length.');
```

```

0057 end
0058 disp(sprintf('** Submitting %d A0s.', na));
0059
0060 %-----
0061 % open mysql connection
0062
0063 conn = mysql_connect();
0064
0065 %-----
0066 % Now process each A0
0067
0068 A0ids = [];
0069 Cid    = -1;
0070
0071 for j=1:na
0072
0073     a = as(j);
0074     % Tag this A0
0075     a = tag(a, conn);
0076
0077     disp(sprintf('** Submitting A0: %06d/%s', a.tag, a.name));
0078
0079     % Get UserID from Username
0080     [userid, dbuser] = mysql_getUserID(conn);
0081     disp(sprintf('** submitting as user %s/%d', dbuser, userid));
0082
0083     % Collect together all info for this entry
0084     name      = a.name;
0085     data      = a.data;
0086     dinfo     = whos('data');
0087     dtype     = dinfo.class;
0088     dname     = data.name;
0089     submitted = ltpda_reformat_date(datestr(now, 31));
0090     prov      = a.provenance;
0091     % created  = ltpda_reformat_date(datestr(get(prov, 'created'), 31));
0092     created   = format(get(prov, 'created'), 'mm-dd-yyyy HH:MM:SS');
```

```

0093 ip          = get(prov, 'ip');
0094 hostname    = get(prov, 'hostname');
0095 os          = get(prov, 'os');
0096 filename    = sprintf('A0%06d.xml', a.tag);
0097
0098 % convert ao to text
0099 x = xml(a); % first to DOM
0100 atxt = xmlwrite(x.docNode); % then to string
0101
0102 % Insert into aos table
0103 colnames = {'name', 'dtype', 'dname', 'ndata', ...
0104             'UserID', 'Comment1', 'Creation', ...
0105             'IP', 'Hostname', 'OS', 'Filename'};
0106 exdata    = {name, dtype, dname, len(a), ...
0107             userid, comment, created, ...
0108             ip, hostname, os, filename};
0109
0110 fastinsert(conn, 'aos', colnames, exdata);
0111
0112 colnames = {'A0'};
0113 exdata    = {atxt};
0114 insert(conn, 'aofiles', colnames, exdata);
0115
0116
0117 % Update transactions table
0118 colnames = {'aoid', 'UserId', 'Tdate', 'Direction'};
0119 exdata    = {a.tag, userid, submitted, 'in'};
0120 try
0121     fastinsert(conn, 'transactions', colnames, exdata);
0122 catch
0123     warning('### error updating transaction table for submission of A0 %s/%d',
a.name, a.tag);
0124 end
0125
0126 % collect submitted IDs
0127 A0ids = [A0ids a.tag];
0128
0129 end
0130
0131 % Update collections table.
0132 colnames = {'num', 'aoids'};
0133 exdata    = {length(A0ids), csv(A0ids)};
0134 if ~isempty(A0ids)
0135     try
0136         fastinsert(conn, 'collections', colnames, exdata);
0137         % get assigned collection ID
0138         curs = exec(conn, 'select max(id) from collections');
0139         curs = fetch(curs);
0140         Cid  = curs.Data{1};
0141         close(curs);
0142     catch
0143         warning('### error submitting A0 collection.');
```



```

0144     end
0145 end
0146
0147 % Cid = [];
0148
0149 %-----
0150 % close MySQL connection
0151 close(conn);
0152
0153 %-----
0154 % make comma separated list of numbers
0155 function s = csv(x)
0156
0157 s = sprintf('%g,', x);
0158 s = s(1:end-1);
0159
0160 %-----
0161 % Get default params
0162 function plo = getDefaultPL()
0163
0164 disp('* creating default plist...');
0165 plo = plist();
0166 disp('* done.');
```

```

0167
0168
0169 %-----
0170 % A faster insert than fastinsert (at least for this case)
0171 function message = mininsert(conn, table, varargin)
0172
0173 % MINSERT inserts values into a single row of a table using JDBC driver
0174 % specified by the input connection.
0175 %
0176 % Usage: message = mininsert(conn, table, 'field1', value1, 'field2',
0177 %                                     value2,...)
0178 %
0179 % M Hewitson
0180 %
0181 % 31-07-07
0182 %
0183 % $Id:$
0184 %
0185
0186 %   q = sprintf('INSERT INTO aofiles(A0) VALUES('%s')';', atxt);
0187 %   curs = exec(conn, q);
0188 %   disp(curs.Message)
0189 %   close(curs);
0190
0191 q = sprintf('INSERT INTO %s(', table);
0192 for j=1:2:length(varargin)
0193     q = [q varargin{j} ','];
0194 end
0195 q = q(1:end-1);
```

```
0196
0197 q = [q ' ) VALUES('];
0198
0199 for j=2:2:length(varargin)
0200     val = varargin{j};
0201     if ischar(val)
0202         q = [q ' ' val ' ' ','];
0203     elseif isnumeric(val)
0204         q = [q mat2str(val) ','];
0205     else
0206         error('unknown data type');
0207     end
0208
0209 end
0210
0211 q = q(1:end-1);
0212 q = [q ');'];
0213
0214 curs = exec(conn, q);
0215 message = curs.Message;
0216 disp(message);
0217 close(curs);
0218
0219
0220 % END
```

[Home](#) > [classes](#) > [@ao](#) > subsasgn.m

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 1.4 2007/11/23 12:11:08 ingo Exp \$

HISTORY: 31-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR overloads char() function for analysis objects.
- [set](#) SET set an analysis object property.

This function is called by:

SOURCE CODE

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for analysis object properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for analysis object properties.
```

```

0007 %
0008 % EXAMPLES:      All possible assignments are possible.
0009 %
0010 % VERSION:       $Id: subsasgn.m,v 1.4 2007/11/23 12:11:08 ingo Exp $
0011 %
0012 % HISTORY:       31-01-07 M Hewitson
0013 %                 Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: subsasgn.m,v 1.4 2007/11/23 12:11:08 ingo Exp $';
0018
0019 % Check if this is a call for parameters or for the cvs-version number
0020 if nargin == 2
0021     if isa(A, 'ao') && ischar(S)
0022         in = char(S);
0023         if strcmp(in, 'Params')
0024             A = plist();
0025             return
0026         elseif strcmp(in, 'Version')
0027             A = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 h = history('set', VERSION, plist(S(end).subs, B));
0034 h = set(h, 'inhists', A.hist);
0035 h = set(h, 'invars', cellstr(inputname(1)));
0036
0037 A.hist = h;
0038
0039 A = builtin ('subsasgn', A, S, B);
0040
0041 % END

```

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 1.14 2007/11/23 12:11:08 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for analysis objects.
0007 %
```

```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:       $Id: subsref.m,v 1.14 2007/11/23 12:11:08 ingo Exp $
0011 %
0012 % HISTORY:       31-01-07 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: subsref.m,v 1.14 2007/11/23 12:11:08 ingo Exp $';
0018
0019 % Check if this is a call for parameters or for the cvs-version number
0020 if nargin == 2
0021     if isa(A, 'ao') && ischar(S)
0022         in = char(S);
0023         if strcmp(in, 'Params')
0024             B = plist();
0025             return
0026         elseif strcmp(in, 'Version')
0027             B = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 B = builtin('subsref', A, S);

```

[Home](#) > [classes](#) > [@ao](#) > testAO.m

testAO

PURPOSE

TESTAO checks that an analysis object has the correct class fields.

SYNOPSIS

```
function testAO(ao)
```

DESCRIPTION

TESTAO checks that an analysis object has the correct class fields.

M Hewitson 01-02-07

\$Id: testAO.m,v 1.2 2007/05/08 10:09:45 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function testAO(ao)
0002
0003 % TESTAO checks that an analysis object has the correct class fields.
0004 %
0005 % M Hewitson 01-02-07
0006 %
0007 % $Id: testAO.m,v 1.2 2007/05/08 10:09:45 hewitson Exp $
0008 %
0009
0010 % check the data field
0011 if ~isa(ao.data, 'tsdata') &&...
0012     ~isa(ao.data, 'fsdata') &&...
```

```
0013     ~isa(ao.data, 'cdata') &&...
0014     ~isa(ao.data, 'xydata') &&...
0015     error('### unknown data object.')
0016 end
0017 % check the history field
0018 if ~isa(ao.hist, 'history')
0019     error('### unknown history object.')
0020 end
0021
0022 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > xml.m

xml

PURPOSE

XML convert an analysis object to an DOM XML object.

SYNOPSIS

function x = xml(a)

DESCRIPTION

XML convert an analysis object to an DOM XML object.

%%%

DESCRIPTION: XML convert an analysis object to an DOM XML object.

CALL: >> x = xml(a)

VERSION: \$Id: xml.m,v 1.8 2007/11/14 16:30:18 ingo Exp \$

HISTORY: 14-02-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add an analysis object to the given node of the input XML DOM

This function is called by:

- [md5](#) MD5 computes an MD5 checksum from an analysis objects.
- [submit_fastinsert](#) SUBMIT AO(s) to the MySQL server.
- [xmladd](#) XMLADD add an analysis object to the given node of the input XML DOM

SOURCE CODE

```
0001 function x = xml(a)
```

```

0002 % XML convert an analysis object to an DOM XML object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert an analysis object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(a)
0009 %
0010 % VERSION: $Id: xml.m,v 1.8 2007/11/14 16:30:18 ingo Exp $
0011 %
0012 % HISTORY: 14-02-2007 M Hewitson
0013 %             Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018 %
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('ao');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now)))));
0023
0024 x = xmladd(a, x, 'Object', x.docRootNode);
0025
0026
0027 % %% Add AO name
0028 % el = x.docNode.createElement('Name');
0029 % el.appendChild(x.docNode.createTextNode(a.name));
0030 % x.docRootNode.appendChild(el);
0031 %
0032 % %% Add data object
0033 % x = xml_add_data(x, a.data);
0034 %
0035 % %% Add history
0036 % x = xml_add_hist(x, a.hist, x.docRootNode);
0037 %
0038 % %% Add AO Provenacne
0039 % x = xml_add_provenance(x, get(a, 'provenance'), x.docRootNode);
0040 %
0041 % %% Add description
0042 % el = x.docNode.createElement('description');
0043 % el.appendChild(x.docNode.createTextNode(get(a, 'description')));
0044 % x.docRootNode.appendChild(el);
0045 %
0046 % %% Add m-file
0047 % el = x.docNode.createElement('M-file');
0048 % el.appendChild(x.docNode.createTextNode(a.mfile));
0049 % x.docRootNode.appendChild(el);
0050 %
0051 % %% Add m-file
0052 % el = x.docNode.createElement('Mfilename');

```

```
0053 % el.appendChild(x.docNode.createTextNode(a.mfilename));
0054 % x.docRootNode.appendChild(el);
0055 %
0056 % %% Add mdl-file
0057 % el = x.docNode.createElement('Mdl-file');
0058 % el.appendChild(x.docNode.createTextNode(a.mdlfile));
0059 % x.docRootNode.appendChild(el);
0060 %
0061 % %% Add mdl-file
0062 % el = x.docNode.createElement('Mdlfilename');
0063 % el.appendChild(x.docNode.createTextNode(a.mdlfilename));
0064 % x.docRootNode.appendChild(el);
0065
0066 % END
```

[Home](#) > [classes](#) > [@ao](#) > xmladd.m

xmladd

PURPOSE

XMLADD add an analysis object to the given node of the input XML DOM

SYNOPSIS

```
function xml = xmladd(as, xml, nodename, node)
```

DESCRIPTION

XMLADD add an analysis object to the given node of the input XML DOM object.

M Hewitson 14-02-07

\$Id: xmladd.m,v 1.4 2007/11/14 16:30:18 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [get](#) GET get ao properties.
- [xml](#) XML convert an analysis object to an DOM XML object.
- [xmladd](#) XMLADD add an analysis object to the given node of the input XML DOM

This function is called by:

- [xml](#) XML convert an analysis object to an DOM XML object.
- [xmladd](#) XMLADD add an analysis object to the given node of the input XML DOM

SOURCE CODE

```
0001 function xml = xmladd(as, xml, nodename, node)
0002
0003 % XMLADD add an analysis object to the given node of the input XML DOM
0004 % object.
```

```

0005 %
0006 % M Hewitson 14-02-07
0007 %
0008 % $Id: xmladd.m,v 1.4 2007/11/14 16:30:18 ingo Exp $
0009 %
0010
0011 el = xml.docNode.createElement(nodename);
0012
0013 for j=1:length(as)
0014     a = as(j);
0015
0016     elc = xml.docNode.createElement('ao');
0017
0018     %% Add AO name
0019     elcc = xml.docNode.createElement('name');
0020     elcc.appendChild(xml.docNode.createTextNode(a.name));
0021     elc.appendChild(elcc);
0022
0023     %% Add data object
0024     if ~isempty(a.data)
0025         xml = xmladd(a.data, xml, 'data', elc);
0026     end
0027
0028     %% Add history
0029     xml = xmladd(a.hist, xml, 'hist', elc);
0030
0031     %% Add AO Provenacne
0032     xml = xmladd(get(a, 'provenance'), xml, 'provenance', elc);
0033
0034     %% Add Description
0035     elcc = xml.docNode.createElement('description');
0036     elcc.appendChild(xml.docNode.createTextNode(get(a, 'description')));
0037     elc.appendChild(elcc);
0038
0039     %% Add m-file
0040     elcc = xml.docNode.createElement('mfile');
0041     elcc.appendChild(xml.docNode.createTextNode(a.mfile));
0042     elc.appendChild(elcc);
0043
0044     %% Add m-file
0045     elcc = xml.docNode.createElement('mfilename');
0046     elcc.appendChild(xml.docNode.createTextNode(a.mfilename));
0047     elc.appendChild(elcc);
0048
0049     %% Add mdl-file
0050

```

```
0051  elcc = xml.docNode.createElement('mdlfile');
0052  elcc.appendChild(xml.docNode.createTextNode(a.mdlfile));
0053  elc.appendChild(elcc);
0054
0055  %% Add mdl-file
0056  elcc = xml.docNode.createElement('mdlfilename');
0057  elcc.appendChild(xml.docNode.createTextNode(a.mdlfilename));
0058  elc.appendChild(elcc);
0059
0060  %% Add AO version
0061  elcc = xml.docNode.createElement('version');
0062  elcc.appendChild(xml.docNode.createTextNode(a.version));
0063  elc.appendChild(elcc);
0064
0065  % add Created
0066  xml = xmladd(as.created, xml, 'created', elc);
0067
0068  el.appendChild(elc);
0069 end
0070
0071 node.appendChild(el);
0072
0073
0074 % END
```

Index for classes/@ao/private

Matlab files in this directory:

 Contents	%% class: ao/private %%
 aoperate	AOPERATE implements specified operator overload for AOs.
 collect_inputs	COLLECT_INPUTS Collect the AOs and Plists.
 fromxml	FROMXML read from part of an XML tree and create an analysis object.
 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

Index for classes/@ao/testing

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% class: ao/testing %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
-  [horzcat](#) HORZCAT overload horizontal concatenation [] for AOs.
-  [vertcat](#) VERTCAT overload horizontal concatenation [] for AOs.

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

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.
 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 vectors in tsdata.
 subsref	SUBSREF Define field name indexing for cdata objects.
 xml	XML convert a cdata object to an DOM XML object.
 xmladd	XMLADD add a cdata object to the given node of the input XML DOM
 xmlparse	XMLPARSE creates a cdata from a cdata xml file.

Subsequent directories:

- [CVS](#)
- [private](#)

[< Master index](#)[Index for classes/@cdata/private >](#)

Index for classes/@cdata/private

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% class: cdata/private %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
-  [fromxml](#) FROMXML read from part of an XML tree and create a cdata object.
-  [retrieve](#) RETRIEVE cdata objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- . CVS

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

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.
 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.
 xml	XML convert an fsdata object to an DOM XML object.
 xmladd	XMLADD add a fsdata object to the given node of the input XML DOM
 xmlparse	XMLPARSE creates a fsdata from a fsdata xml file.

Subsequent directories:

- [CVS](#)
- [private](#)

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

[<](#) Master indexIndex for classes/@fsdata/private [>](#)

Index for classes/@fsdata/private

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% class: fsdata/private %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
-  [fromxml](#) FROMXML read from part of an XML tree and create a fsdata object.
-  [retrieve](#) RETRIEVE fsdata objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- . CVS

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

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.
 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.
 xml	XML convert a specwin object to an DOM XML object.
 xmladd	XMLADD add an history object to the given node of the input XML DOM
 xmlparse	XMLPARSE creates a history from a history xml file.

Subsequent directories:

- CVS
- [private](#)

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

[< Master index](#)[Index for classes/@history/private >](#)

Index for classes/@history/private

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% class: history/private %%%%%%%%%%
 fromxml	FROMXML read from part of an XML tree and create a history object.
 retrieve	RETRIEVE history objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- . CVS

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

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.
 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.
 xml	XML convert an mfir object to an DOM XML object.
 xmladd	XMLADD add an MFIR object to the given node of the input XML DOM
 xmlparse	XMLPARSE creates a mfir filter from a mfir xml file.

Subsequent directories:

- [CVS](#)
- [private](#)

Index for classes/@mfir/private

Matlab files in this directory:

 Contents	%% class: mfir/private %%%%%%%%%%%%%%%%%%%%%%%%%%
 fromxml	FROMXML read from part of an XML tree and create an mfir object.
 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](#)

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.
 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.
 xml	XML convert an miir object to an DOM XML object.
 xmladd	XMLADD add an MIIR object to the given node of the input XML DOM
 xmlparse	XMLPARSE creates a miir filter from a miir xml file.

Subsequent directories:

- [CVS](#)
- [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
 fromxml	FROMXML read from part of an XML tree and create an miir object.
 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

[< Master index](#)[Index for classes/@param/private >](#)

Index for classes/@param/private

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% class: param/private %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
-  [fromxml](#) FROMXML read from part of an XML tree and create a param object.
-  [retrieve](#) RETRIEVE param objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- . CVS

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

Index for classes/@plist/private

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% class: plist/private %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
-  [fromxml](#) FROMXML read from part of an XML tree and create a plist object.
-  [retrieve](#) RETRIEVE plist objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- . CVS

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

Index for classes/@pole/private

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% class: pole/private %%%%%%%%%%%%%%
 fromxml	FROMXML read from part of an XML tree and create a zero object.
 pfq2ri	Convert frequency/Q pole/zero representation into real
 pri2fq	Convert complex 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 22-Jan-2008 10:39:01 by [m2html](#) © 2003

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.
 find_real_ip	FIND_REAL_IP finds the real ip address and not the localhost address (127.0.0.1)
 get	GET get provenance properties.
 isfield	ISFIELD tests if the given field is one of the object properties.
 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.
 subsasgn	SUBSASGN define index assignment for provenance properties.
 subsref	SUBSREF Define field name indexing for provenance objects.
 xml	XML convert a provenance object to an DOM XML object.
 xmladd	XMLADD adds specwin object to the given node of the xml input
 xmlparse	XMLPARSE creates a provenance from a provenance xml file.

Subsequent directories:

- [CVS](#)
- [private](#)

[< Master index](#)[Index for classes/@provenance/private >](#)

Index for classes/@provenance/private

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% class: provenance/private %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
-  [fromxml](#) FROMXML read from part of an XML tree and create a provenance object.
-  [retrieve](#) RETRIEVE provenance objects with given ids from the LTPDA repository
specified by the input

Subsequent directories:

- . CVS

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

Index for classes/@pzmodel/private

Matlab files in this directory:

 Contents	%% class: pzmodel/private %%%%%%%%%%%%%%%%%%%%%%%%% %%
 abccascade	Cascade two filters together to get a new filter.
 fromxml	FROMXML read from part of an XML tree and create a pzmodel object.
 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 22-Jan-2008 10:39:01 by [m2html](#) © 2003

[< Master index](#)[Index for classes/@specwin/private >](#)

Index for classes/@specwin/private

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% class: specwin/private %%%%%%%%%%
 fromxml	FROMXML read from part of an XML tree and create a specwin object.
 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

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.
 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.
 xml	XML convert a time object to an DOM XML object.
 xmladd	XMLADD add a time object to the given node of the input XML DOM
 xmlparse	XMLPARSE creates a time from a time xml file.

Subsequent directories:

- [CVS](#)
- [private](#)

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

[< Master index](#)[Index for classes/@time/private >](#)

Index for classes/@time/private

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% class: time/private %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
-  [fromxml](#) FROMXML read from part of an XML tree and create a time object.
-  [retrieve](#) RETRIEVE time objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- . CVS

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

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.
 isfield	ISFIELD tests if the given field is one of the object properties.
 ne	NE overloads the ~= operator for timeformat objects.
 save	SAVE a timeformat object to file.
 set	SET set timeformat properties.
 subsasgn	SUBSASGN define index assignment for timeformat properties.
 subsref	SUBSREF Define field name indexing for timeformat objects.
 timeformat	TIMEFORMAT timeformat object class constructor.
 xml	XML convert a timeformat object to an DOM XML object.
 xmladd	XMLADD add a time object to the given node of the input XML DOM
 xmlparse	XMLPARSE creates a timeformat from a timeformat xml file.

Subsequent directories:

- [CVS](#)
- [private](#)

[< Master index](#)[Index for classes/@timeformat/private >](#)

Index for classes/@timeformat/private

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% class: timeformat/private %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
-  [fromxml](#) FROMXML read from part of an XML tree and create a timeformat object.
-  [retrieve](#) RETRIEVE timeformat objects with given ids from the LTPDA repository
specified by the input

Subsequent directories:

- . CVS

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

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.
 isfield	ISFIELD tests if the given field is one of the object properties.
 ne	NE overloads the ~= operator for timespan objects.
 save	SAVE a timespan object to file.
 set	SET set timespan properties.
 subsasgn	SUBSASGN define index assignment for timespan properties.
 subsref	SUBSREF Define field name indexing for timespan objects.
 timespan	TIMESPAN timespan object class constructor.
 xml	XML convert a time object to an DOM XML object.
 xmladd	XMLADD add a timespan object to the given node of the input XML DOM
 xmlparse	XMLPARSE creates a timespan from a timespan xml file.

Subsequent directories:

- [CVS](#)
- [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.

 [fromxml](#) FROMXML read from part of an XML tree and create a timespan object.

 [retrieve](#) RETRIEVE timespan objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- CVS

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

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.
 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.
 xml	XML convert a tsdata object to an DOM XML object.
 xmladd	XMLADD add a tsdata object to the given node of the input XML DOM
 xmlparse	XMLPARSE creates a tsdata from a tsdata xml file.

Subsequent directories:

- [CVS](#)
- [private](#)

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

[< Master index](#)[Index for classes/@tsdata/private >](#)

Index for classes/@tsdata/private

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% class: tsdata/private %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
-  [fromxml](#) FROMXML read from part of an XML tree and create a tsdata object.
-  [retrieve](#) RETRIEVE tsdata objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- . CVS

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

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.
 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.
 xml	XML convert an xydata object to an DOM XML object.
 xmladd	XMLADD add a xydata object to the given node of the input XML DOM
 xmlparse	XMLPARSE creates a xydata from a xydata xml file.
 xydata	XYDATA X-Y data object class constructor.

Subsequent directories:

- [CVS](#)
- [private](#)

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

[< Master index](#)[Index for classes/@xydata/private >](#)

Index for classes/@xydata/private

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% class: xydata/private %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
-  [fromxml](#) FROMXML read from part of an XML tree and create a xydata object.
-  [retrieve](#) RETRIEVE xydata objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

- . CVS

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

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.
 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

Index for classes/@zero/private

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% class: zero/private %%%%%%%%%%%%%%% %%%%%%%%%%%%%%
 fromxml	FROMXML read from part of an XML tree and create a zero object.
 pfq2ri	Convert frequency/Q pole/zero representation into real
 pri2fq	Convert complex 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 22-Jan-2008 10:39:01 by [m2html](#) © 2003

Index for m/analyses/stuttgart

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% path: m/analyses/stuttgart %%%% %%%%%%%%%%%%%%
 dkalman_matrices	%%%%%%%%%%%%%% %%%%%%%%%%%%%%
 e2e2ao	%%%%%%%%%%%%%% %%%%%%%%%%%%%%
 ltpda_diff_acc	LTPDA_DIFF_ACC
 ltpda_dkalman	LTPDA_DKALMAN
 test_dkalman	%%%%%%%%%%%%%% %%%%%%%%%%%%%%

Other Matlab-specific files in this directory:

- input_meas_kalm.mat

Subsequent directories:

- CVS

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

Subsequent directories:

- [CVS](#)

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 22-Jan-2008 10:39:01 by [m2html](#) © 2003

Index for m/gui/ltpdaRepoGUI

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% path: m/gui/ltpdaRepoGUI %%%%%%%%%%
 mk_test_objs	
 muitable	UITABLE creates a two dimensional graphicuitable 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.
 uitabdemo	UITABDEMO This script demonstrates the use of UITABPANEL.
 uitabpanel	UITABPANEL adds a group of tabbed uipanel container objects to a figure.

Subsequent directories:

- [CVS](#)
- [callbacks](#)

[<](#) Master indexIndex for m/gui/ltpdaRepoGUI/callbacks [>](#)

Index for m/gui/ltpdaRepoGUI/callbacks

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% path: m/gui/ ltpdaRepoGUI/callbacks %%%%%%%%%%%%%%% %
 buildConditions	
 buildquery	
 cb_classMethodList	
 cb_condPlusBtn	
 cb_condSubtractBtn	
 cb_connectBtn	
 cb_dbSelect	
 cb_executeQuery	Get connection
 cb_fieldSelect	Build query
 cb_getDBs	
 cb_mainfigClose	
 cb_refreshObjsList	get a list of objects
 cb_refreshObjsListQL	get a list of objects
 cb_retrieveBtn	Get connection
 cb_serverSelect	
 cb_submitBtn	
 cb_tableSelect	
 cb_workspaceObjsList	
 cb_workspaceObjsListQL	
 fillClassMethodsList	Get the object types
 getFields	get the selected table
 getTables	
 getWorkspaceObjs	get base workspace variables
 setWorkspaceObjsList	

 [setWorkspaceObjsListQL](#)

Subsequent directories:

- [CVS](#)

Generated on Tue 22-Jan-2008 10:39:01 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 22-Jan-2008 10:39:01 by [m2html](#) © 2003

Index for m/helper

Matlab files in this directory:

 [Contents](#)

 [dunzip](#)

 [dzip](#)

 [gcbsh](#)

 [get_curr_m_file_path](#)

 [ltpda_deg2rad](#)

 [ltpda_fields2list](#)

 [ltpda_filescan](#)

 [ltpda_fq2ri](#)

 [ltpda_fq2ri2](#)

 [ltpda_funchash](#)

 [ltpda_getmax](#)

 [ltpda_hash](#)

 [ltpda_io_collect](#)

 [ltpda_isobject](#)

 [ltpda_isuserobject](#)

 [ltpda_label](#)

 [ltpda_mat2str](#)

%%%%%%%%%%%%%% path: m/helper %%%%
%%%%%%%%%%%%%%

DUNZIP - decompress DZIP output to recover original data

DZIP - losslessly compress data into smaller memory space

GCBSH gets the handles for the currently selected blocks.

GET_CURR_M_FILE_PATH returns the path for a mfile.

LTPDA_DEG2RAD Convert degrees to radians

LTPDA_FIELDS2LIST splits a string containing fields
seperated by ','

LTPDA_FILESCAN recursively scans the given directory for
files that end in 'ext'

LTPDA_FQ2RI Convert frequency/Q pole/zero
representation into real

LTPDA_FQ2RI2 Convert frequency/Q pole/zero
representation into real

LTPDA_FUNCHASH compute MD5 hash of an LTPDA
function.

LTPDA_GETMAX get the index and value of the maximum
value in an array.

LTPDA_HASH - Convert an input variable into a message
digest using any of

LTPDA_IO_COLLECT collects all AOs and plists from the
inputs.

LTPDA_ISOBJECT checks that the input objects are one of
the LTPDA object

LTPDA_ISUSEROBJECT checks that the input objects are
one of the LTPDA user object

LTPDA_LABEL makes the input string into a suitable string
for using on plots.

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_xml_read_datasamples](#)

LTPDA_XML_READ_DATASAMPLES read data samples from the given node of an XML file.

 [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

[< 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 22-Jan-2008 10:39:01 by [m2html](#) © 2003

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_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

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	% get zeros and poles from pzmodels
 nginit	
 ngprop	
 ngsetup	ALGONAME = mfilename;

Subsequent directories:

- [CVS](#)
- [private](#)

Generated on Tue 22-Jan-2008 10:39:01 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](#)

 [fq2fac](#)

Subsequent directories:

- [CVS](#)

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

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](#)

[< Master index](#)[Index for m/sigproc/frequency_domain >](#)

Index for m/sigproc/frequency_domain

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% path: m/sigproc/frequency_domain %%%%%%%%%%%%%%
 ltpda_cohere	LTPDA_COHERE makes coherence estimates of the time-series objects
 ltpda_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	LTPDA_LTFE implement weighted 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_xspect	LTPDA_XSPEC performs cross-spectral analysis of various forms.

Subsequent directories:

- [CVS](#)

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

Index for m/sigproc/time_domain

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% path: m/sigproc/time_domain %%%%%%%%%%%%%%
 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_polydetrend	LTPDA_POLYDETREND detrends the input analysis object using a polynomial of degree N.
 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

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

Generated on Tue 22-Jan-2008 10:39:01 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

Subsequent directories:

- [CVS](#)
- [macintel](#)
- [windows](#)

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

[< Master index](#)

[Index for mex/macintel/2007b >](#)

Index for mex/macintel/2007b

Matlab files in this directory:

 [ltpda_dft](#) LTPDA_DFT computes the DFT of a signal at one frequency.

Subsequent directories:

- [CVS](#)

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

[<](#) Master index

Index for mex/windows/2007b [>](#)

Index for mex/windows/2007b

Matlab files in this directory:



[ltpda_dft](#) LTPDA_DFT computes the DFT of a signal at one frequency.

Generated on Tue 22-Jan-2008 10:39:01 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@provenance](#) > find_real_ip.m

find_real_ip

PURPOSE

FIND_REAL_IP finds the real ip address and not the localhost address (127.0.0.1)

SYNOPSIS

function output_ip_address = find_real_ip(varargin)

DESCRIPTION

FIND_REAL_IP finds the real ip address and not the localhost address (127.0.0.1)

%%%

DESCRIPTION: find_real_ip finds the real ip address and not the localhost address (127.0.0.1)

CALL: ip = find_real_ip();

VERSION: \$Id: find_real_ip.m,v 1.2 2008/01/11 14:00:08 ingo Exp \$

HISTORY: 04-01-2008 Diepholz
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- **char** CHAR convert a provenance object into a string.

This function is called by:

SOURCE CODE

```
0001 function output_ip_address = find_real_ip(varargin)
0002 % FIND_REAL_IP finds the real ip address and not the localhost address
0003 % (127.0.0.1)
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
```

```

0006 % DESCRIPTION: find_real_ip finds the real ip address and not the localhost
0007 %                  address (127.0.0.1)
0008 %
0009 % CALL:           ip = find_real_ip();
0010 %
0011 % VERSION:        $Id: find_real_ip.m,v 1.2 2008/01/11 14:00:08 ingo Exp $
0012 %
0013 % HISTORY:        04-01-2008 Diepholz
0014 %                  Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018
0019 output_ip_address = NaN;
0020
0021 import java.net.*;
0022
0023 interfaces = NetworkInterface.getNetworkInterfaces();
0024
0025 while(interfaces.hasMoreElements())
0026
0027     ni = interfaces.nextElement();
0028
0029     if ni.isLoopback == 1, continue, end
0030
0031     ip_list = ni.getInterfaceAddresses().listIterator();
0032
0033     while(ip_list.hasNext())
0034
0035         ip = ip_list.next();
0036
0037         ip_address = ip.getAddress();
0038
0039         if (~ip_address.isLoopbackAddress && ip_address.getHostAddress().indexOf
('::') == -1)
0040             output_ip_address = ip_address.getHostAddress;
0041             disp(sprintf('IP Address [%s] loopback [%d] SiteLocal [%d]', char
(ip_address.getHostAddress), ip_address.isLoopbackAddress(), ip_address.
isSiteLocalAddress())) ;
0042         end
0043
0044     end
0045 end
0046

```

[Home](#) > [m](#) > [sigproc](#) > [frequency_domain](#) > ltpda_xspec.m

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

```

0001 function varargout = ltpda_xspec(varargin)
0002 % LTPDA_XSPEC performs cross-spectral analysis of various forms.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_XSPEC performs cross-spectral analysis of various forms.
0007 %               The function is a helper function for various higher level
0008 %               functions. It is meant to be called from other functions
0009 %               (e.g., ltpda_tfe).
0010 %
0011 % CALL:         b = ltpda_xspec(a, pl, method, iALGO, iVER, invars);
0012 %
0013 % INPUTS:       a      - vector of input AOs
0014 %               pl     - input parameter list
0015 %               method - one of 'TF', 'CPSD', 'COHERE'
0016 %               iALGO  - ALGONAME from the calling higher level script
0017 %               iVER   - VERSION from the calling higher level script
0018 %               invars - invars variable from the calling higher level script
0019 %
0020 % OUTPUTS:      b      - Na x Na matrix of output AOs
0021 %
0022 % VERSION:      $Id: ltpda_xspec.m,v 1.10 2007/11/08 10:24:54 hewitson Exp $
0023 %
0024 % HISTORY:      30-05-2007 M Hewitson
0025 %               Creation
0026 %
0027 % The following call returns a parameter list object that contains the
0028 % default parameter values:
0029 %
0030 % >> pl = ltpda_xspec('Params')
0031 %
0032 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0033 %
0034 % unpack inputs
0035 as      = varargin{1};

```

```

0036 pl      = varargin{2};
0037 method = varargin{3};
0038 iALGO    = varargin{4};
0039 iVER      = varargin{5};
0040 invars    = varargin{6};
0041
0042 ALGONAME = iALGO;
0043 VERSION  = [iVER '/' '$Id: ltpda_xspec.m,v 1.10 2007/11/08 10:24:54 hewitson
Exp $'];
0044
0045
0046 % initialise output array
0047 bs = [];
0048
0049 na = length(as);
0050 if na < 2
0051     error('### LTPDA_XSPEC needs at least two AOs to make a transfer function.');
```

0052 end

0053

0054

0055 %----- Resample all AOs

0056

0057 disp('*** Resampling AOs...');

0058 fsmax = findFsMax(as);

0059 fspl = plist(param('fsout', fsmax));

0060 bs = [];

0061 for i=1:na

0062 a = as(i);

0063 % check this is a time-series object

0064 if ~isa(a.data, 'tsdata')

0065 error('### ltpda_xspec requires tsdata (time-series) inputs.');

0066 end

0067 % Check Fs

0068 if a.data.fs ~= fsmax

0069 warning(sprintf('!!! Resampling AO %s/%s to %f Hz', a.name, a.data.name,

fsmax));

0070 a = resample(a, fspl);

0071 end

0072 bs = [bs a];

0073 end

0074 as = bs;

0075 clear bs;

0076 disp('*** Done.');

0077

0078

0079 %----- Truncate all vectors

0080

0081 % Get shortest vector

0082 disp('*** Truncating all vectors...');

0083 lmin = findShortestVector(as);

0084 nsecs = lmin / fsmax;

0085 bs = [];

0086 for i=1:na

```

0087     a = as(i);
0088     if len(a) ~= lmin
0089         warning(sprintf('!!! Truncating A0 %s/%s to %d secs', a.name, a.data.name,
nsecs));
0090         bs = [bs select(a, 1:lmin)];
0091     else
0092         bs = [bs a];
0093     end
0094 end
0095 as = bs;
0096 clear bs;
0097 disp('*** Done.');
```

0098

```

0099 %----- check input parameters
0100
0101 % points in FFT
0102 Nfft = find(pl, 'Nfft');
0103 if ~isempty(Nfft)
0104     if Nfft < 0
0105         Nfft = abs(Nfft) * lmin;
0106         disp(sprintf('! Using default Nfft of %g', Nfft))
0107         pl = pset(pl, 'Nfft', Nfft);
0108     end
0109 end
0110
0111 % Window object
0112 Win = find(pl, 'Win');
0113 if ~isempty(Win)
0114     if length(Win.win)~=Nfft
0115         switch Win.name
0116             case 'Kaiser'
0117                 Win = specwin(Win.name, Nfft, Win.psll);
0118             otherwise
0119                 Win = specwin(Win.name, Nfft);
0120         end
0121         disp(sprintf('! Reset window to %s(%d)', strrep(Win.name, '_', '\_'), length
(Win.win)))
0122         pl = pset(pl, 'Win', Win);
0123     end
0124 end
0125
0126 % desired overlap
0127 Noverlap = find(pl, 'Noverlap');
0128 if ~isempty(Noverlap)
0129     if Noverlap < 0
0130         % use window function rov
0131         Noverlap = Win.rov/100;
0132         disp(sprintf('! Using recommended overlap of %d', Noverlap))
0133         pl = pset(pl, 'Noverlap', Noverlap);
0134     end
0135 end
0136
0137 % desired detrending order
```

```

0138 order = find(pl, 'Order');
0139 if isempty(order)
0140     order = 0;
0141     warning('! using default detrending order 0 (mean)');
0142     pl = append(pl, 'Order', order);
0143 end
0144
0145 % Loop over input AOs
0146 bs = cell(na);
0147 for j=1:na
0148     aj = as(j);
0149     for k=1:na
0150         ak = as(k);
0151         % ----- Make Xspec estimate
0152
0153         % Do detrending
0154         x = aj.data.y;
0155         y = ak.data.y;
0156         switch order
0157             case -1
0158                 % do nothing
0159             case 0
0160                 x = x - mean(x);
0161                 y = y - mean(y);
0162             case 1
0163                 x = detrend(x);
0164                 y = detrend(y);
0165             otherwise
0166                 y = polydetrend(y, order);
0167                 y = polydetrend(y, order);
0168         end
0169
0170         switch method
0171             case 'TF'
0172                 % Compute TF using tfestimate
0173                 [txy,f] = tfestimate(x,y,Win.win,round(Nolap*Nfft),Nfft,fsmax);
0174             case 'COHERE'
0175                 % Compute coherence using mscohere
0176                 [txy,f] = mscohere(x,y,Win.win,round(Nolap*Nfft),Nfft,fsmax);
0177             case 'CPSD'
0178                 % Compute Cross-spectral density using CPSD
0179                 [txy,f] = cpsd(x,y,Win.win,round(Nolap*Nfft),Nfft,fsmax);
0180             otherwise
0181                 error('### Unknown method.')
0182         end
0183
0184         % create new output fsdata
0185         fs = fsdata(f, txy, fsmax);
0186         fs = set(fs, 'name', sprintf('%s(%s->%s)', method, aj.data.name, ak.data.
name));
0187         fs = set(fs, 'enbw', Win.nenbw*fsmax/Nfft);
0188
0189         %----- create new output history

```

```

0190
0191 % we need to get the input histories in the same order as the inputs
0192 % to this function call, not in the order of the input to tfestimate;
0193 % otherwise the resulting matrix on a 'create from history' will be
0194 % mirrored.
0195 if j<k
0196     h = history(ALGONAME, VERSION, pl, [aj.hist ak.hist]);
0197 else
0198     h = history(ALGONAME, VERSION, pl, [ak.hist aj.hist]);
0199 end
0200 if na == length(invars)
0201     h = set(h, 'invars', [invars(j) invars(k)]);
0202     nk = invars{k};
0203     nj = invars{j};
0204 else
0205     nk = ak.name;
0206     nj = aj.name;
0207 end
0208
0209 % make output analysis object
0210 b = ao(fs, h);
0211
0212 % set name
0213 b = setnh(b, 'name', sprintf('%s(%s->%s)', method, nj, nk));
0214
0215 % add to outputs
0216 bs(j,k) = {b};
0217 end % End second loop over AOs
0218 end % End first loop over AOs
0219
0220 % now we have a cell matrix of AOs but
0221 % we want a normal matrix
0222 b = [bs{:}];
0223 varargout{1} = reshape(b, na, na);
0224
0225
0226 %-----
0227 % Returns the length of the shortest vector in samples
0228 function lmin = findShortestVector(as)
0229
0230 lmin = 1e20;
0231 for j=1:length(as)
0232     if len(as(j)) < lmin
0233         lmin = len(as(j));
0234     end
0235 end
0236
0237
0238 %-----
0239 % Returns the max Fs of a set of AOs
0240 function fs = findFsMax(as)
0241
0242 fs = 0;

```



```
0243 for j=1:length(as)
0244     a = as(j);
0245     if a.data.fs > fs
0246         fs = a.data.fs;
0247     end
0248 end
0249
0250
0251
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > [private](#) > Contents.m

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:

[Home](#) > [classes](#) > [@time](#) > format.m

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.

%%%

DESCRIPTION: FORMAT Returns the time in the handover format.

CALL: string = format(tt, format_str);
 string = format(tt, 'HH:MM:SS');

VERSION: \$Id: format.m,v 1.6 2008/01/14 20:03:52 ingo Exp \$

HISTORY: 23-07-2007 Diepholz
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a time object into a string.
- [set](#) SET set time properties.
- [string](#) STRING writes a command string that can be used to recreate the input time object.

This function is called by:

- [plus](#) PLUS overloads - operator for time objects.

SOURCE CODE

```
0001 function string = format(tt, format_str)
0002 % FORMAT Returns the time in the handover format.
```

```

0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: FORMAT Returns the time in the handover format.
0007 %
0008 % CALL:          string = format(tt, format_str);
0009 %               string = format(tt, 'HH:MM:SS');
0010 %
0011 % VERSION:       $Id: format.m,v 1.6 2008/01/14 20:03:52 ingo Exp $
0012 %
0013 % HISTORY:       23-07-2007 Diepholz
0014 %                 Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: format.m,v 1.6 2008/01/14 20:03:52 ingo Exp $';
0019
0020 % Check if this is a call for parameters
0021 if nargin == 2
0022     if isa(tt, 'time') && ischar(format_str)
0023         in = char(format_str);
0024         if strcmp(in, 'Params')
0025             string = plist;
0026             return
0027         elseif strcmp(in, 'Version')
0028             string = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 default_t = set(tt, 'timeformat', tt.timeformat.default_matlab_str);
0035
0036 curr_time_number = datenum(default_t.time_str, default_t.timeformat.format_str);
0037
0038 string = datestr (curr_time_number, format_str);
0039

```

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

```
0001 function ltpda_xticks(v)
0002 % LTPDA_XTICKS set the input vector as the y-ticks of the current axis.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_XTICKS set the input vector as the y-ticks of the current
0007 %                axis.
0008 %
```

```
0009 % CALL:      ltpda_xticks(v)
0010 %
0011 % VERSION:    $Id: ltpda_xticks.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
0012 %
0013 % HISTORY:     10-08-2006 M Hewitson
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%
0017
0018 set(gca, 'Xtick', v);
0019
0020 % END
```

[Home](#) > [classes](#) > [@cdata](#) > single_operation.m

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.

%%%

CROSS-REFERENCE INFORMATION [^](#)

This function calls:

- [char](#) CHAR convert a cdata object into a string.

This function is called by:

SUBFUNCTIONS [^](#)

- [function pl_default = getDefaultPL\(\)](#)

SOURCE CODE [^](#)

```

0001 function varargout = single_operation (varargin)
0002 % SINGLE_OPERATION implements specified operator overload for cdata objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SINGLE_OPERATION implements specified operator overload
0007 %               for cdata objects.
0008 %               ATTENTION: Should only be called from a analysis function.
0009 %
0010 % CALL: [h, data_out] = single_operation(a.data, 'svd', pl)
0011 %
0012 % varargin format: 1:   [cdata]
0013 %                  2:   ['operation']
0014 %                  3:   [plist]
0015 %
0016 % varargout format 1: [new_history]
0017 %                  2: [data_out1]
0018 %                  3: [data_out2]      optional
0019 %                  4: [data_out3]      optional
0020 %                  ...                optional
0021 %
0022 % With the plist values you can control the command line.
0023 % Possible key values are: option    <double> or <char>
0024 %                        dim        <double> or <char>
0025 %                        W          <vector>
0026 %
0027 % >> operation(data,'option')          option <char>
0028 % >> operation(data,option)            option <double>
0029 % >> operation(data,W,dim)
0030 % >> operation(data,dim)
0031 % >> operation(data,W)
0032 % >> operation(data)

```

```

0033 %
0034 % HISTORY: 07-05-2007 Diepholz
0035 %           Creation.
0036 %
0037 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0038
0039 VERSION = '$Id: single_operation.m,v 1.13 2007/11/26 18:07:51 ingo Exp $';
0040
0041 % Check if this is a call for parameters
0042 if nargin == 2
0043     if isa(varargin{1}, 'cdata') && ischar(varargin{2})
0044         in = char(varargin{2});
0045         if strcmp(in, 'Params')
0046             varargout{1} = getDefaultPL();
0047             return
0048         elseif strcmp(in, 'Version')
0049             varargout{1} = VERSION;
0050             return
0051         end
0052     end
0053 end
0054
0055 % Only the call from an analysis object is allowed.
0056 if (nargin == 3)
0057
0058     data = varargin {1};
0059     op    = varargin {2};
0060     pl    = varargin {3};
0061     a     = varargin {end};
0062     y     = data.y;
0063
0064 %% Verify input variables and set default parameter
0065 if ~isa(data, 'cdata')
0066     error('### cdata/single_operation: The first input must be a cdata.');
```

```

0067 end
0068 if ~ischar(op)
0069     error('### cdata/single_operation: The second input must be a string.');
```

```

0070 end
0071 if ~isa(plist, 'plist')
0072     error('### cdata/single_operation: The third input must be a plist.');
```

```

0073 end
0074
0075 %% Set default parameter list
0076 pl_default = getDefaultPL();
0077
0078 %% Combine the handover param-list and the default param-list
0079 if isempty(pl)
0080     pl = combine(pl);
0081 else
0082     pl = combine(pl_default);
0083 end
0084

```

```

0085 option      = find(pl, 'option');
0086 dim          = find(pl, 'dim');
0087 weight_vector = find(pl, 'w');
0088 do_xdata      = find(pl, 'xdata');
0089 do_ydata      = find(pl, 'ydata');
0090
0091 %% Check the <char> option if it begins and ends with a quote '
0092 %% If not then add the quote.
0093 if ischar(option) && ~isempty(option)
0094     if ~strcmp(option(1), '')
0095         option = ['' option];
0096     end
0097     if ~strcmp(option(end), '')
0098         option = [option ''];
0099     end
0100 end
0101
0102 %% Convert the <double> option into a char
0103 if isnumeric(option) || islogical(option)
0104     option = num2str(option);
0105 end
0106
0107 %% Compute always the y data
0108 do_xdata = 'x';
0109 do_ydata = 'y';
0110
0111 %% Convert the dim into a char
0112 if isnumeric(dim) || islogical(dim)
0113     dim = num2str(dim);
0114 end
0115
0116 %% The operation can only compute with a 'option' or with a 'dimension'.
0117 %% e.g. operation(ao, option)
0118 %%      operation(ao, dim)
0119 if ~isempty(option) && ~isempty(dim)
0120     warning('### xydata/single_operation: The param list contain a ''option''
and a ''dim''. It is computed %s(ao,option).', op);
0121 end
0122
0123 %% %% Ensure that the called function have at least two output variables
0124 if (nargout >= 2)
0125
0126 %% %% Create the output string for the eval command
0127 %% e.g.: y_new = '[ y_new1 y_new2 ]'
0128 y_new = '[';
0129 for i = 1:nargout-1
0130     new_var_name = genvarname(sprintf('y_new%d', i));
0131     eval([new_var_name '= 0;']);
0132
0133     y_new = strcat(y_new, sprintf(' y_new%d', i));
0134 end
0135 y_new = strcat(y_new, ' ]');

```

```

0136
0137 %% %% Create the output variables
0138 %% e.g.: data_out1 = data;
0139 for i = 1:nargout-1
0140     new_var_name = genvarname (sprintf('data_out%d',i));
0141     eval ([new_var_name '= data;']);
0142 end
0143
0144 %% Execute the command
0145
0146 try
0147
0148     %% Execute with a option or a dimension
0149     %% >> operation(y,'option');
0150     if ~isempty(option)
0151         cmd_line = sprintf('%s = %s(y,%s);',y_new, op, option);
0152         eval (cmd_line);
0153
0154     %% Execute with a weight vector and dimension
0155     %% >> operation(y,W,dim);
0156     elseif ~isempty(dim) && ~isempty(weight_vector)
0157         cmd_line = sprintf('%s = %s(y,weight_vector,%s);',y_new, op, dim);
0158         eval (cmd_line);
0159
0160     %% Execute with a option or a dimension
0161     %% >> operation(y,dim);
0162     elseif ~isempty(dim)
0163         cmd_line = sprintf('%s = %s(y,%s);',y_new, op, dim);
0164         eval (cmd_line);
0165
0166     %% Execute with a weight vector
0167     %% >> operation(y,W);
0168     elseif ~isempty(weight_vector)
0169         cmd_line = sprintf('%s = %s(y,weight_vector);',y_new, op);
0170         eval (cmd_line);
0171
0172     %% Execute without parameter or option
0173     %% >> operation(y)
0174     else
0175         cmd_line = sprintf('%s = %s(y);',y_new, op);
0176         eval (cmd_line);
0177     end
0178
0179 catch
0180     l_error = lasterror;
0181     disp(sprintf('\n\n%s', l_error.message));
0182     error('### cdata/single_operation: Can not command:\n###      %s\n### Have
a look to the param list', cmd_line);
0183 end
0184
0185
0186 %% %% The result from eval is in y_new1, ...

```

```

0187     %% Create a cdata object from this results and write it into the output
0188     for i = 1:nargout-1
0189
0190         %% Create cdata
0191         eval (sprintf('data_out%d.y = y_new%d;',i,i));
0192         eval (sprintf('data_out%d.name = '%s(%s)'';',i,op,data.name));
0193
0194         %% Write to output
0195         eval (sprintf('varargout{i+1} = data_out%d;',i))
0196
0197     end
0198
0199     %% Create a new history object
0200     h = history(op, VERSION);
0201
0202     varargout{1} = h;
0203
0204     else % (nargout < 2)
0205         error ('### cdata/single_operation: This function need at least two output
variables. ');
0206     end
0207
0208 else
0209     error ('### cdata/single_operation: This function need three input
variables. ')
0210 end
0211
0212
0213 function pl_default = getDefaultPL()
0214
0215     pl_default = plist('option', '', ...
0216                       'dim',    '', ...
0217                       'w',      [], ...
0218                       'xdata',  'x', ...
0219                       'ydata',  'y');
0220

```

[Home](#) > [classes](#) > [@ao](#) > [testing](#) > Contents.m

Contents

PURPOSE

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: ao/testing %%%%%%%%%%
```

SYNOPSIS

This is a script file.

DESCRIPTION

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: ao/testing %%%%%%%%%%

    <a href="matlab:help ao/testing/horzcat">ao/testing/horzcat</a> -  overload
horizontal concatonation [] for AOs.
    <a href="matlab:help ao/testing/vertcat">ao/testing/vertcat</a> -  overload
horizontal concatonation [] for AOs.
```

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [noisegenerator](#) > [private](#) > fq2fac.m

fq2fac

PURPOSE

SYNOPSIS

```
function polzero = fq2fac(f,q)
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function polzero = fq2fac(f,q)
0002
0003 n = length(f);
0004 polzero = zeros(n,3);
0005 for i = 1:n
0006     if isnan(q(i))
0007         polzero(i,1:2) = [1 1/(2*pi*f(i))];
0008     else
0009         polzero(i,1:3) = [1 1/(2*pi*f(i)*q(i)) 1/((2*pi*f(i))*(2*pi*f
(i)))]];
0010     end
0011 end
```

Generated on Tue 22-Jan-2008 10:39:13 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

```
0001 function ltpda_yaxis(y1,y2)
0002 % LTPDA_YAXIS Set the Y axis range of the current figure
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_YAXIS Set the Y axis range of the current figure
0007 %
0008 % CALL:      ltpda_yaxis(x1,x2)
0009 %
```



```
0010 % VERSION:      $Id: ltpda_yaxis.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
0011 %
0012 % HISTORY:       09-10-2001 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 set(gca, 'YLim', [y1 y2]);
0018
0019 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@fsdata](#) > single_operation.m

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.
```

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

CROSS-REFERENCE INFORMATION

This function calls:

- `char` CHAR convert a param object into a string.

This function is called by:

SUBFUNCTIONS

- `function pl_default = getDefaultPL()`

SOURCE CODE

```
0001 function varargout = single_operation (varargin)
0002 % SINGLE_OPERATION implements specified operator overload for fsdata objects.
0003 %
0004 %%%%%%%%%%
0005 %
0006 % DESCRIPTION: SINGLE_OPERATION implements specified operator overload
0007 %               for fsdata objects.
0008 %               ATTENTION: Should only be called from a analysis function.
0009 %
0010 % CALL: [h, data_out] = single_operation(a.data, 'log', pl)
0011 %
0012 % varargin format: 1:   [fsdata]
0013 %                  2:   ['operation']
0014 %                  3:   [plist]
0015 %
0016 % varargout format 1: [new_history]
0017 %                  2: [data_out1]
0018 %                  3: [data_out2]   optional
0019 %                  4: [data_out3]   optional
0020 %                  ...             optional
0021 %
0022 % With the plist values you can control the command line.
0023 % Possible key values are: option  <double> or <char>
0024 %                          dim     <double> or <char>
0025 %                          W       <vector>
0026 %                          xdata   <char>
0027 %                          ydata   <char>
0028 % With the keys xdata and ydata you can control which data are to compute.
0029 % Attention the value of the key 'xdata' and 'ydata' must apply to the data
```

```

class.
0030 % The other values are controle the command line:
0031 %
0032 % >> operation(data,'option')           option <char>
0033 % >> operation(data,option)             option <double>
0034 % >> operation(data,W,dim)
0035 % >> operation(data,dim)
0036 % >> operation(data,W)
0037 % >> operation(data)
0038 %
0039 % HISTORY: 07-05-2007 Diepholz
0040 %           Creation.
0041 %
0042 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0043
0044 VERSION = '$Id: single_operation.m,v 1.11 2007/11/27 13:19:19 ingo Exp $';
0045
0046 % Check if this is a call for parameters
0047 if nargin == 2
0048     if isa(varargin{1}, 'fsdata') && ischar(varargin{2})
0049         in = char(varargin{2});
0050         if strcmp(in, 'Params')
0051             varargout{1} = getDefaultPL();
0052             return
0053         elseif strcmp(in, 'Version')
0054             varargout{1} = VERSION;
0055             return
0056         end
0057     end
0058 end
0059
0060 % Only the call from an analysis object is allowed.
0061 if (nargin == 3)
0062
0063     data = varargin {1};
0064     op    = varargin {2};
0065     pl    = varargin {3};
0066     a     = varargin {end};
0067     y     = data.y;
0068     x     = data.x;
0069
0070 %% Verify input variables and set default parameter
0071 if ~isa(data, 'fsdata')
0072     error ('### fsdata/single_operation: The first input must be a fsdata.');
```

```

0082   pl_default = getDefaultPL();
0083
0084   %% Combine the handover param-list and the default param-list
0085   if ~isempty(pl)
0086       pl = combine(pl);
0087   else
0088       pl = combine(pl_default);
0089   end
0090
0091   option          = find(pl, 'option');
0092   dim              = find(pl, 'dim');
0093   weight_vector    = find(pl, 'w');
0094   do_xdata         = find(pl, 'xdata');
0095   do_ydata         = find(pl, 'ydata');
0096
0097   %% Check the <char> option if it begins and ends with a quote '
0098   %% If not then add the quote.
0099   if ischar(option) && ~isempty(option)
0100       if ~strcmp(option(1), '')
0101           option = ['' option];
0102       end
0103       if ~strcmp(option(end), '')
0104           option = [option ''];
0105       end
0106   end
0107
0108   %% Convert the <double> option into a char
0109   if isnumeric(option) || islogical(option)
0110       option = num2str(option);
0111   end
0112
0113   %% Convert the dim into a char
0114   if isnumeric(dim) || islogical(dim)
0115       dim = num2str(dim);
0116   end
0117
0118   %% The operation can only compute with a 'option' or with a 'dimension'.
0119   %% e.g. operation(ao, 'option')
0120   %%      operation(ao, dim)
0121   if ~isempty(option) && ~isempty(dim)
0122       warning('### xydata/single_operation: The param list contain a ''option''
and a ''dim''. It is computed %s(ao,option).', op);
0123   end
0124
0125   %%
0126   if isempty(do_xdata) && isempty(do_ydata)
0127       do_ydata = 'y';
0128   end
0129
0130   %% %% Ensure that the called function have at least two output variables
0131   if (nargout >= 2)
0132
0133   %% %% y (y-axis):

```

```

0134 %% Create the output string for the eval command
0135 %% e.g.: y_new = '[ y_new1 y_new2 ]'
0136 y_new = '[';
0137 for i = 1:nargout-1
0138     new_var_name = genvarname (sprintf('y_new%d',i));
0139     eval ([new_var_name '= 0;']);
0140
0141     y_new = strcat(y_new, sprintf (' y_new%d',i));
0142 end
0143 y_new = strcat (y_new, ' ');
0144
0145 %% %% t (x-axis):
0146 %% Create the output string for the eval command
0147 %% e.g.: x_new = '[ x_new1 x_new2 ]'
0148 x_new = '[';
0149 for i = 1:nargout-1
0150     new_var_name = genvarname (sprintf('x_new%d',i));
0151     eval ([new_var_name '= 0;']);
0152
0153     x_new = strcat(x_new, sprintf (' x_new%d',i));
0154 end
0155 x_new = strcat (x_new, ' ');
0156
0157 %% %% Create the output variables
0158 %% e.g.: data_out1 = data;
0159 for i = 1:nargout-1
0160     new_var_name = genvarname (sprintf('data_out%d',i));
0161     eval ([new_var_name '= data;']);
0162 end
0163
0164 %% Execute the command for y (y-axis)
0165 if ~isempty(do_ydata)
0166
0167     try
0168         %% Execute with a option
0169         %% >> operation(y,'option');
0170         if ~isempty(option)
0171             cmd_line = sprintf('%s = %s(y,%s);',y_new, op, option);
0172             eval (cmd_line);
0173
0174             %% Execute with a weight vector and dimension
0175             %% >> operation(y,W,dim);
0176             elseif ~isempty(dim) && ~isempty(weight_vector)
0177                 cmd_line = sprintf('%s = %s(y,weight_vector,%s);',y_new, op, dim);
0178                 eval (cmd_line);
0179
0180             %% Execute with a dimension
0181             %% >> operation(y,dim);
0182             elseif ~isempty(dim)
0183                 cmd_line = sprintf('%s = %s(y,%s);',y_new, op, dim);
0184                 eval (cmd_line);
0185
0186             %% Execute with a weight vector

```

```

0187     %% >> operation(y,W);
0188     elseif ~isempty(weight_vector)
0189         cmd_line = sprintf('%s = %s(y,weight_vector);',y_new, op);
0190         eval (cmd_line);
0191
0192     %% Execute without parameter or option
0193     %% >> operation(y)
0194     else
0195         cmd_line = sprintf('%s = %s(y);',y_new, op);
0196         eval (cmd_line);
0197     end
0198
0199     catch
0200         l_error = lasterror;
0201         disp(sprintf('\n\n%s', l_error.message));
0202         error('### cdata/single_operation: Can not command:\n###      %s\n###
Have a look to the param list', cmd_line);
0203     end
0204
0205 end
0206
0207 %% Execute the command for x (x-axis)
0208 if ~isempty(do_xdata)
0209
0210     try
0211         %% Execute with a option or a dimension
0212         %% >> operation(x,'option');
0213         if ~isempty(option)
0214             cmd_line = sprintf('%s = %s(x,%s);',x_new, op, option);
0215             eval (cmd_line);
0216
0217         %% Execute with a weight vector and dimension
0218         %% >> operation(x,W,dim);
0219         elseif ~isempty(dim) && ~isempty(weight_vector)
0220             cmd_line = sprintf('%s = %s(x,weight_vector,%s);',x_new, op, dim);
0221             eval (cmd_line);
0222
0223         %% Execute with a option or a dimension
0224         %% >> operation(x,dim);
0225         elseif ~isempty(dim)
0226             cmd_line = sprintf('%s = %s(x,%s);',x_new, op, dim);
0227             eval (cmd_line);
0228
0229         %% Execute with a weight vector
0230         %% >> operation(x,W);
0231         elseif ~isempty(weight_vector)
0232             cmd_line = sprintf('%s = %s(x,weight_vector);',x_new, op);
0233             eval (cmd_line);
0234
0235         %% Execute without parameter or option
0236         %% >> operation(x)
0237         else
0238             cmd_line = sprintf('%s = %s(x);',x_new, op);

```

```

0239         eval (cmd_line);
0240     end
0241
0242     catch
0243         l_error = lasterror;
0244         disp(sprintf('\n\n%s', l_error.message));
0245         error('### cdata/single_operation: Can not command:\n###      %s\n###
Have a look to the param list', cmd_line);
0246     end
0247
0248 end
0249
0250 %% %% The result from eval is in y_new1, x_new1, ...
0251 %% Create a fsdata object from this results and write it into the output
0252 for i = 1:nargout-1
0253
0254     %% Create fsdata
0255     %% y data (y-axis)
0256     if ~isempty(do_ydata)
0257         eval (sprintf('data_out%d.y = y_new%d;', i, i));
0258         eval (sprintf('data_out%d.name = '%s(%s)'';', i, op, data.name));
0259         eval (sprintf('data_out%d.yunits = '%s(%s)'';', i, op, data.yunits));
0260     end
0261     %% t data (x-axis)
0262     if ~isempty(do_xdata)
0263         eval (sprintf('data_out%d.x = x_new%d;', i, i));
0264         eval (sprintf('data_out%d.name = '%s(%s)'';', i, op, data.name));
0265         eval (sprintf('data_out%d.xunits = '%s(%s)'';', i, op, data.xunits));
0266     end
0267
0268     %% Write to output
0269     eval (sprintf('varargout{i+1} = data_out%d;', i))
0270
0271 end
0272
0273 %% Create a new history object
0274 h = history(op, VERSION);
0275
0276 varargout{1} = h;
0277
0278 else % (nargout < 2)
0279     error ('### fsdata/single_operation: This function need at least two output
variables. ');
0280 end
0281
0282 else
0283     error ('### fsdata/single_operation: This function need three input
variables. ')
0284 end
0285
0286
0287 function pl_default = getDefaultPL()
0288

```



```
0289     pl_default = plist('option', '', ...
0290                        'dim',    '', ...
0291                        'w',      [], ...
0292                        'xdata',  '', ...
0293                        'ydata',  'y');
0294
```

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 cdata
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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create an analysis object.

SYNOPSIS

function a = fromxml(xml)

DESCRIPTION

FROMXML read from part of an XML tree and create an analysis object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.4 2007/11/14 16:30:18 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE analysis objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function a = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create an analysis object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.4 2007/11/14 16:30:18 ingo Exp $
0008 %
0009
0010 % defaults
```

```

0011 a = ao;
0012
0013 sch = xml.getChildNodes;
0014 for c=1:sch.getLength
0015     x = sch.item(c-1);
0016     switch char(x.getNodeName)
0017         case 'name'
0018             a = setnh(a, 'name', char(x.getTextContent));
0019         case 'data'
0020             xch = x.getChildNodes;
0021             dtype = char(xch.item(1).getNodeName);
0022             switch dtype
0023                 case 'tsdata'
0024                     a = setnh(a, 'data', tsdata(xch.item(1)));
0025                 case 'fsdata'
0026                     a = setnh(a, 'data', fsdata(xch.item(1)));
0027                 case 'xydata'
0028                     a = setnh(a, 'data', xydata(xch.item(1)));
0029                 case 'cdata'
0030                     a = setnh(a, 'data', cdata(xch.item(1)));
0031             end
0032         case 'hist'
0033             xch = x.getChildNodes;
0034             a = setnh(a, 'hist', history(xch.item(1)));
0035         case 'provenance'
0036             xch = x.getChildNodes;
0037             a = setnh(a, 'provenance', provenance(xch.item(1)));
0038         case 'description'
0039             a = setnh(a, 'description', char(x.getTextContent));
0040         case 'mfile'
0041             a = setnh(a, 'mfile', ltpda_str2cells(char(x.getTextContent)));
0042         case 'mfilename'
0043             a = setnh(a, 'mfilename', char(x.getTextContent));
0044         case 'mdlfile'
0045             a = setnh(a, 'mdlfile', char(x.getTextContent));
0046         case 'mdlfilename'
0047             a = setnh(a, 'mdlfilename', char(x.getTextContent));
0048         case 'version'
0049             a = setnh(a, 'version', char(x.getTextContent));
0050         case 'created'
0051             xch = x.getChildNodes;
0052             a = setnh(a, 'created', time(xch.item(1)));
0053         otherwise
0054     end
0055 end
0056
0057

```

0058 % END

Generated on Tue 22-Jan-2008 10:39:13 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

```
0001 function ltpda_yscale(scale)
0002 % LTPDA_YSCALE Set the Y scale of the current axis
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_YSCALE Set the Y scale of the current axis
0007 %
0008 % CALL:          ltpda_yscale('scale')          scale = 'lin' or 'log';
0009 %
```

```
0010 % VERSION:      $Id: ltpda_yscale.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
0011 %
0012 % HISTORY:       02-05-2004 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 set(gca, 'YScale', scale);
0018
0019 % END
```


[Home](#) > [classes](#) > [@tsdata](#) > single_operation.m

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.
```

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

CROSS-REFERENCE INFORMATION

This function calls:

- **char** CHAR convert a param object into a string.

This function is called by:

SUBFUNCTIONS

- **function** pl_default = getDefaultPL()

SOURCE CODE

```
0001 function varargout = single_operation (varargin)
0002 % SINGLE_OPERATION implements specified operator overload for tsdata objects.
0003 %
0004 %%%%%%%%%%
0005 %
0006 % DESCRIPTION: SINGLE_OPERATION implements specified operator overload
0007 %               for tsdata objects.
0008 %               ATTENTION: Should only be called from a analysis function.
0009 %
0010 % CALL: [h, data_out] = single_operation(a.data, 'log', pl)
0011 %
0012 % varargin format: 1:   [tsdata]
0013 %                  2:   ['operation']
0014 %                  3:   [plist]
0015 %
0016 % varargout format 1: [new_history]
0017 %                  2: [data_out1]
0018 %                  3: [data_out2]   optional
0019 %                  4: [data_out3]   optional
0020 %                  ...             optional
0021 %
0022 % With the plist values you can control the command line.
0023 % Possible key values are: option   <double> or <char>
0024 %                        dim       <double> or <char>
0025 %                        W         <vector>
0026 %                        xdata     <char>
0027 %                        ydata     <char>
0028 % With the keys xdata and ydata you can control which data are to compute.
0029 % Attention the value of the key 'xdata' and 'ydata' must apply to the data
```

```

class.
0030 % The other values are controle the command line:
0031 %
0032 % >> operation(data,'option')           option <char>
0033 % >> operation(data,option)             option <double>
0034 % >> operation(data,W,dim)
0035 % >> operation(data,dim)
0036 % >> operation(data,W)
0037 % >> operation(data)
0038 %
0039 % HISTORY: 07-05-2007 Diepholz
0040 %           Creation.
0041 %
0042 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0043
0044 VERSION = '$Id: single_operation.m,v 1.11 2007/11/27 13:19:19 ingo Exp $';
0045
0046 % Check if this is a call for parameters
0047 if nargin == 2
0048     if isa(varargin{1}, 'tsdata') && ischar(varargin{2})
0049         in = char(varargin{2});
0050         if strcmp(in, 'Params')
0051             varargout{1} = getDefaultPL();
0052             return
0053         elseif strcmp(in, 'Version')
0054             varargout{1} = VERSION;
0055             return
0056         end
0057     end
0058 end
0059
0060 % Only the call from an analysis object is allowed.
0061 if (nargin == 3)
0062
0063     data = varargin {1};
0064     op   = varargin {2};
0065     pl   = varargin {3};
0066     a    = varargin {end};
0067     y    = data.y;
0068     x    = data.x;
0069
0070 %% Verify input variables and set default parameter
0071 if ~isa(data, 'tsdata')
0072     error ('### tsdata/single_operation: The first input must be a tsdata.');
```

```

0082   pl_default = getDefaultPL();
0083
0084   %% Combine the handover param-list and the default param-list
0085   if ~isempty(pl)
0086       pl = combine(pl);
0087   else
0088       pl = combine(pl_default);
0089   end
0090
0091   option          = find(pl, 'option');
0092   dim              = find(pl, 'dim');
0093   weight_vector   = find(pl, 'w');
0094   do_xdata         = find(pl, 'xdata');
0095   do_ydata        = find(pl, 'ydata');
0096
0097   %% Check the <char> option if it begins and ends with a quote '
0098   %% If not then add the quote.
0099   if ischar(option) && ~isempty(option)
0100       if ~strcmp(option(1), ''')
0101           option = [''' option];
0102       end
0103       if ~strcmp(option(end), ''')
0104           option = [option '''];
0105       end
0106   end
0107
0108   %% Convert the <double> option into a char
0109   if isnumeric(option) || islogical(option)
0110       option = num2str(option);
0111   end
0112
0113   %% Convert the dim into a char
0114   if isnumeric(dim) || islogical(dim)
0115       dim = num2str(dim);
0116   end
0117
0118   %% The operation can only compute with a 'option' or with a 'dimension'.
0119   %% e.g. operation(ao, 'option')
0120   %%      operation(ao, dim)
0121   if ~isempty(option) && ~isempty(dim)
0122       warning('### xydata/single_operation: The param list contain a ''option''
and a ''dim''. It is computed %s(ao,option).', op);
0123   end
0124
0125   %%
0126   if isempty(do_xdata) && isempty(do_ydata)
0127       do_ydata = 'y';
0128   end
0129
0130   %% %% Ensure that the called function have at least two output variables
0131   if (nargout >= 2)
0132
0133   %% %% y (y-axis):

```

```

0134 %% Create the output string for the eval command
0135 %% e.g.: y_new = '[ y_new1 y_new2 ]'
0136 y_new = '[';
0137 for i = 1:nargout-1
0138     new_var_name = genvarname (sprintf('y_new%d',i));
0139     eval ([new_var_name '= 0;']);
0140
0141     y_new = strcat(y_new, sprintf (' y_new%d',i));
0142 end
0143 y_new = strcat (y_new, ' ');
0144
0145 %% %% x (x-axis):
0146 %% Create the output string for the eval command
0147 %% e.g.: x_new = '[ x_new1 x_new2 ]'
0148 x_new = '[';
0149 for i = 1:nargout-1
0150     new_var_name = genvarname (sprintf('x_new%d',i));
0151     eval ([new_var_name '= 0;']);
0152
0153     x_new = strcat(x_new, sprintf (' x_new%d',i));
0154 end
0155 x_new = strcat (x_new, ' ');
0156
0157 %% %% Create the output variables
0158 %% e.g.: data_out1 = data;
0159 for i = 1:nargout-1
0160     new_var_name = genvarname (sprintf('data_out%d',i));
0161     eval ([new_var_name '= data;']);
0162 end
0163
0164 %% Execute the command for y (y-axis)
0165 if ~isempty(do_ydata)
0166
0167     try
0168         %% Execute with a option
0169         %% >> operation(y,'option');
0170         if ~isempty(option)
0171             cmd_line = sprintf('%s = %s(y,%s);',y_new, op, option);
0172             eval (cmd_line);
0173
0174             %% Execute with a weight vector and dimension
0175             %% >> operation(y,W,dim);
0176             elseif ~isempty(dim) && ~isempty(weight_vector)
0177                 cmd_line = sprintf('%s = %s(y,weight_vector,%s);',y_new, op, dim);
0178                 eval (cmd_line);
0179
0180             %% Execute with a dimension
0181             %% >> operation(y,dim);
0182             elseif ~isempty(dim)
0183                 cmd_line = sprintf('%s = %s(y,%s);',y_new, op, dim);
0184                 eval (cmd_line);
0185
0186             %% Execute with a weight vector

```

```

0187     %% >> operation(y,W);
0188     elseif ~isempty(weight_vector)
0189         cmd_line = sprintf('%s = %s(y,weight_vector);',y_new, op);
0190         eval (cmd_line);
0191
0192     %% Execute without parameter or option
0193     %% >> operation(y)
0194     else
0195         cmd_line = sprintf('%s = %s(y);',y_new, op);
0196         eval (cmd_line);
0197     end
0198
0199     catch
0200         l_error = lasterror;
0201         disp(sprintf('\n\n%s', l_error.message));
0202         error('### cdata/single_operation: Can not command:\n###      %s\n###
Have a look to the param list', cmd_line);
0203     end
0204
0205 end
0206
0207 %% Execute the command for x (x-axis)
0208 if ~isempty(do_xdata)
0209
0210     try
0211         %% Execute with a option or a dimension
0212         %% >> operation(x,'option');
0213         if ~isempty(option)
0214             cmd_line = sprintf('%s = %s(x,%s);',x_new, op, option);
0215             eval (cmd_line);
0216
0217         %% Execute with a weight vector and dimension
0218         %% >> operation(x,W,dim);
0219         elseif ~isempty(dim) && ~isempty(weight_vector)
0220             cmd_line = sprintf('%s = %s(x,weight_vector,%s);',x_new, op, dim);
0221             eval (cmd_line);
0222
0223         %% Execute with a option or a dimension
0224         %% >> operation(x,dim);
0225         elseif ~isempty(dim)
0226             cmd_line = sprintf('%s = %s(x,%s);',x_new, op, dim);
0227             eval (cmd_line);
0228
0229         %% Execute with a weight vector
0230         %% >> operation(x,W);
0231         elseif ~isempty(weight_vector)
0232             cmd_line = sprintf('%s = %s(x,weight_vector);',x_new, op);
0233             eval (cmd_line);
0234
0235         %% Execute without parameter or option
0236         %% >> operation(x)
0237         else
0238             cmd_line = sprintf('%s = %s(x);',x_new, op);

```

```

0239         eval (cmd_line);
0240     end
0241
0242     catch
0243         l_error = lasterror;
0244         disp(sprintf('\n\n%s', l_error.message));
0245         error('### cdata/single_operation: Can not command:\n###      %s\n###
Have a look to the param list', cmd_line);
0246     end
0247
0248 end
0249
0250 %% %% The result from eval is in y_new1, x_new1, ...
0251 %% Create a tsdata object from this results and write it into the output
0252 for i = 1:nargout-1
0253
0254     %% Create tsdata
0255     %% y data (y-axis)
0256     if ~isempty(do_ydata)
0257         eval (sprintf('data_out%d.y = y_new%d;', i, i));
0258         eval (sprintf('data_out%d.name = '%s(%s)'';', i, op, data.name));
0259         eval (sprintf('data_out%d.yunits = '%s(%s)'';', i, op, data.yunits));
0260     end
0261     %% x data (x-axis)
0262     if ~isempty(do_xdata)
0263         eval (sprintf('data_out%d.x = x_new%d;', i, i));
0264         eval (sprintf('data_out%d.name = '%s(%s)'';', i, op, data.name));
0265         eval (sprintf('data_out%d.xunits = '%s(%s)'';', i, op, data.xunits));
0266     end
0267
0268     %% Write to output
0269     eval (sprintf('varargout{i+1} = data_out%d;', i))
0270
0271 end
0272
0273 %% Create a new history object
0274 h = history(op, VERSION);
0275
0276 varargout{1} = h;
0277
0278 else % (nargout < 2)
0279     error ('### tsdata/single_operation: This function need at least two output
variables. ');
0280 end
0281
0282 else
0283     error ('### tsdata/single_operation: This function need three input
variables. ')
0284 end
0285
0286 function pl_default = getDefaultPL()
0287
0288     pl_default = plist('option', '', ...

```

```
0289         'dim',      '',      ...  
0290         'w',      [],      ...  
0291         'xdata',   '',      ...  
0292         'ydata',   'y');  
0293
```


[Home](#) > [classes](#) > [@cdata](#) > [private](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@cdata](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create a cdata object.

SYNOPSIS

```
function data = fromxml(xml)
```

DESCRIPTION

FROMXML read from part of an XML tree and create a cdata object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.2 2007/11/09 14:07:39 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE cdata objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function data = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create a cdata object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.2 2007/11/09 14:07:39 ingo Exp $
```

```

0008 %
0009
0010 % defaults
0011 xunits = '';
0012 yunits = '';
0013 dtype = '';
0014
0015 sch = xml.getChildNodes;
0016 for c=1:sch.getLength
0017     x = sch.item(c-1);
0018     switch char(x.getNodeName)
0019         case 'name'
0020             name = char(x.getTextContent);
0021         case 'xunits'
0022             xunits = char(x.getTextContent);
0023         case 'yunits'
0024             yunits = char(x.getTextContent);
0025         case 'version'
0026             version = char(x.getTextContent);
0027         case 'created'
0028             xch = x.getChildNodes;
0029             created = time(xch.item(1));
0030         case 'xset'
0031             xset = ltpda_str2cells(char(x.getTextContent));
0032         case 'shape'
0033             shape = str2num(char(x.getTextContent));
0034         case 'dtype'
0035             dtype = char(x.getTextContent);
0036         case 'data'
0037             idata = ltpda_xml_read_datasamples(x, dtype);
0038         otherwise
0039     end
0040 end
0041
0042 % build cdata
0043 data = cdata(reshape(idata.y, shape));
0044
0045 data = set(data, 'name', name);
0046 data = set(data, 'version', version);
0047 data = set(data, 'created', created);
0048 data = set(data, 'xunits', xunits);
0049 data = set(data, 'yunits', yunits);
0050 data = set(data, 'x', xset);
0051

```

0052

0053 % END

Generated on Tue 22-Jan-2008 10:39:13 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

```
0001 function ltpda_yticks(v)
0002 % LTPDA_YTICKS set the input vector as the y-ticks of the current axis.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_YTICKS set the input vector as the y-ticks of the current
0007 %                axis.
0008 %
```

```
0009 % CALL:      ltpda_yticks(v)
0010 %
0011 % VERSION:    $Id: ltpda_yticks.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
0012 %
0013 % HISTORY:    10-08-2006 M Hewitson
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%
0017
0018 set(gca, 'Ytick', v);
0019
0020 % END
```

[Home](#) > [classes](#) > [@xydata](#) > single_operation.m

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.

%%%

DESCRIPTION: **SINGLE_OPERATION** implements specified operator overload
for xydata objects.
ATTENTION: Should only be called from a analysis function.

CALL: [h, data_out] = single_operation(a.data, 'log', pl)

```
varargin format: 1:  [xydata]
                  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.
```

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

CROSS-REFERENCE INFORMATION

This function calls:

- **char** CHAR convert a param object into a string.

This function is called by:

SUBFUNCTIONS

- **function pl_default = getDefaultPL()**

SOURCE CODE

```
0001 function varargout = single_operation (varargin)
0002 % SINGLE_OPERATION implements specified operator overload for xydata objects.
0003 %
0004 %%%%%%%%%%  
0005 %
0006 % DESCRIPTION: SINGLE_OPERATION implements specified operator overload
0007 %               for xydata objects.
0008 %               ATTENTION: Should only be called from a analysis function.
0009 %
0010 % CALL: [h, data_out] = single_operation(a.data, 'log', pl)
0011 %
0012 % varargin format: 1:   [xydata]
0013 %                  2:   ['operation']
0014 %                  3:   [plist]
0015 %
0016 % varargout format 1: [new_history]
0017 %                  2: [data_out1]
0018 %                  3: [data_out2]   optional
0019 %                  4: [data_out3]   optional
0020 %                  ...              optional
0021 %
0022 % With the plist values you can control the command line.
0023 % Possible key values are: option   <double> or <char>
0024 %                        dim       <double> or <char>
0025 %                        W         <vector>
0026 %                        xdata     <char>
0027 %                        ydata     <char>
0028 % With the keys xdata and ydata you can control which data are to compute.
0029 % Attention the value of the key 'xdata' and 'ydata' must apply to the data
```



```

class.
0030 % The other values are controle the command line:
0031 %
0032 % >> operation(data,'option')           option <char>
0033 % >> operation(data,option)             option <double>
0034 % >> operation(data,W,dim)
0035 % >> operation(data,dim)
0036 % >> operation(data,W)
0037 % >> operation(data)
0038 %
0039 % HISTORY: 07-05-2007 Diepholz
0040 %           Creation.
0041 %
0042 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0043
0044 VERSION = '$Id: single_operation.m,v 1.11 2007/11/27 13:19:19 ingo Exp $';
0045
0046 % Check if this is a call for parameters
0047 if nargin == 2
0048     if isa(varargin{1}, 'xydata') && ischar(varargin{2})
0049         in = char(varargin{2});
0050         if strcmp(in, 'Params')
0051             varargout{1} = getDefaultPL();
0052             return
0053         elseif strcmp(in, 'Version')
0054             varargout{1} = VERSION;
0055             return
0056         end
0057     end
0058 end
0059
0060 % Only the call from an analysis object is allowed.
0061 if (nargin == 3)
0062
0063     data = varargin {1};
0064     op    = varargin {2};
0065     pl    = varargin {3};
0066     a     = varargin {end};
0067     x     = data.x;
0068     y     = data.y;
0069
0070 %% Verify input variables and set default parameter
0071 if ~isa(data, 'xydata')
0072     error ('### xydata/single_operation: The first input must be a xydata.');
```

```

0082   pl_default = getDefaultPL();
0083
0084   %% Combine the handover param-list and the default param-list
0085   if ~isempty(pl)
0086       pl = combine(pl);
0087   else
0088       pl = combine(pl_default);
0089   end
0090
0091   option          = find(pl, 'option');
0092   dim             = find(pl, 'dim');
0093   weight_vector   = find(pl, 'w');
0094   do_xdata        = find(pl, 'xdata');
0095   do_ydata        = find(pl, 'ydata');
0096
0097   %% Check the <char> option if it begins and ends with a quote '
0098   %% If not then add the quote.
0099   if ischar(option) && ~isempty(option)
0100       if ~strcmp(option(1), '''')
0101           option = ['''' option];
0102       end
0103       if ~strcmp(option(end), '''')
0104           option = [option '''];
0105       end
0106   end
0107
0108   %% Convert the <double> option into a char
0109   if isnumeric(option) || islogical(option)
0110       option = num2str(option);
0111   end
0112
0113   %% Convert the dim into a char
0114   if isnumeric(dim) || islogical(dim)
0115       dim = num2str(dim);
0116   end
0117
0118   %% The operation can only compute with a 'option' or with a 'dimension'.
0119   %% e.g. operation(ao, 'option')
0120   %%      operation(ao, dim)
0121   if ~isempty(option) && ~isempty(dim)
0122       warning('### xydata/single_operation: The param list contain a ''option''
and a ''dim''. It is computed %s(ao,option).', op);
0123   end
0124
0125   %%
0126   if isempty(do_xdata) && isempty(do_ydata)
0127       do_ydata = 'y';
0128   end
0129
0130   %% %% Ensure that the called function have at least two output variables
0131   if (nargout >= 2)
0132
0133   %% %% xdata:

```

```

0134 %% Create the output string for the eval command
0135 %% e.g.: x_new = '[ x_new1 x_new2 ]'
0136 x_new = '[';
0137 for i = 1:nargout-1
0138     new_var_name = genvarname (sprintf('x_new%d',i));
0139     eval ([new_var_name '= 0;']);
0140
0141     x_new = strcat(x_new, sprintf (' x_new%d',i));
0142 end
0143 x_new = strcat (x_new, ' ');
0144
0145 %% ydata:
0146 %% Create the output string for the eval command
0147 %% e.g.: y_new = '[ y_new1 y_new2 ]'
0148 y_new = '[';
0149 for i = 1:nargout-1
0150     new_var_name = genvarname (sprintf('y_new%d',i));
0151     eval ([new_var_name '= 0;']);
0152
0153     y_new = strcat(y_new, sprintf (' y_new%d',i));
0154 end
0155 y_new = strcat (y_new, ' ');
0156
0157 %% %% Create the output variables
0158 %% e.g.: data_out1 = data;
0159 for i = 1:nargout-1
0160     new_var_name = genvarname (sprintf('data_out%d',i));
0161     eval ([new_var_name '= data;']);
0162 end
0163
0164 %% Execute the command for the xdata
0165 if ~isempty(do_xdata)
0166
0167     try
0168         %% Execute with a option
0169         %% >> operation(x,'option');
0170         if ~isempty(option)
0171             cmd_line = sprintf('%s = %s(x,%s);',x_new, op, option);
0172             eval (cmd_line);
0173
0174             %% Execute with a weight vector and dimension
0175             %% >> operation(x,W,dim);
0176             elseif ~isempty(dim) && ~isempty(weight_vector)
0177                 cmd_line = sprintf('%s = %s(x,weight_vector,%s);',x_new, op, dim);
0178                 eval (cmd_line);
0179
0180             %% Execute with a dimension
0181             %% >> operation(x,dim);
0182             elseif ~isempty(dim)
0183                 cmd_line = sprintf('%s = %s(x,%s);',x_new, op, dim);
0184                 eval (cmd_line);
0185
0186             %% Execute with a weight vector

```

```

0187     %% >> operation(x,W);
0188     elseif ~isempty(weight_vector)
0189         cmd_line = sprintf('%s = %s(x,weight_vector);',x_new, op);
0190         eval (cmd_line);
0191
0192     %% Execute without parameter or option
0193     %% >> operation(x)
0194     else
0195         cmd_line = sprintf('%s = %s(x);',x_new, op);
0196         eval (cmd_line);
0197     end
0198
0199     catch
0200         l_error = lasterror;
0201         disp(sprintf('\n\n%s', l_error.message));
0202         error('### cdata/single_operation: Can not command:\n###      %s\n###
Have a look to the param list', cmd_line);
0203     end
0204
0205 end
0206
0207 %% Execute the command for the ydata
0208 if ~isempty(do_ydata)
0209
0210     try
0211         %% Execute with a option or a dimension
0212         %% >> operation(y,'option');
0213         if ~isempty(option)
0214             cmd_line = sprintf('%s = %s(y,%s);',y_new, op, option);
0215             eval (cmd_line);
0216
0217         %% Execute with a weight vector and dimension
0218         %% >> operation(y,W,dim);
0219         elseif ~isempty(dim) && ~isempty(weight_vector)
0220             cmd_line = sprintf('%s = %s(y,weight_vector,%s);',y_new, op, dim);
0221             eval (cmd_line);
0222
0223         %% Execute with a option or a dimension
0224         %% >> operation(y,dim);
0225         elseif ~isempty(dim)
0226             cmd_line = sprintf('%s = %s(y,%s);',y_new, op, dim);
0227             eval (cmd_line);
0228
0229         %% Execute with a weight vector
0230         %% >> operation(y,W);
0231         elseif ~isempty(weight_vector)
0232             cmd_line = sprintf('%s = %s(y,weight_vector);',y_new, op);
0233             eval (cmd_line);
0234
0235         %% Execute without parameter or option
0236         %% >> operation(y)
0237         else
0238             cmd_line = sprintf('%s = %s(y);',y_new, op);

```

```

0239         eval (cmd_line);
0240     end
0241
0242     catch
0243         l_error = lasterror;
0244         disp(sprintf('\n\n%s', l_error.message));
0245         error('### cdata/single_operation: Can not command:\n###      %s\n###
Have a look to the param list', cmd_line);
0246     end
0247
0248 end
0249
0250 %% %% The result from eval is in x_new1, y_new1, ...
0251 %% Create a xydata object from this results and write it into the output
0252 for i = 1:nargout-1
0253
0254     %% Create xydata
0255     if ~isempty(do_xdata)
0256         eval (sprintf('data_out%d.x = x_new%d;', i, i));
0257         eval (sprintf('data_out%d.name = '%s(%s)'';', i, op, data.name));
0258         eval (sprintf('data_out%d.xunits = '%s(%s)'';', i, op, data.xunits));
0259     end
0260     if ~isempty(do_ydata)
0261         eval (sprintf('data_out%d.y = y_new%d;', i, i));
0262         eval (sprintf('data_out%d.name = '%s(%s)'';', i, op, data.name));
0263         eval (sprintf('data_out%d.yunits = '%s(%s)'';', i, op, data.yunits));
0264     end
0265
0266     %% Write to output
0267     eval (sprintf('varargout{i+1} = data_out%d;', i))
0268
0269 end
0270
0271 %% Create a new history object
0272 h = history(op, VERSION);
0273
0274 varargout{1} = h;
0275
0276 else % (nargout < 2)
0277     error ('### xydata/single_operation: This function need at least two output
variables. ');
0278 end
0279
0280 else
0281     error ('### xydata/single_operation: This function need three input
variables. ')
0282 end
0283
0284 function pl_default = getDefaultPL()
0285
0286 pl_default = plist('option', '', ...
0287                   'dim',    '', ...
0288                   'w',      [], ...

```

```
0289         'xdata', ' ', ...  
0290         'ydata', 'y');  
0291
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

Contents

PURPOSE

%% class: fsdata %%%

SYNOPSIS

This is a script file.

DESCRIPTION

%% class: fsdata %%%

fsdata/char	- convert a param object into a string.
fsdata/display	- implement terminal display for fsdata object.
fsdata/eq	- overloads the == operator for fsdata objects.
fsdata/fsdata	- frequency-series object class constructor.
fsdata/get	- get a fsdata property.
fsdata/get_xy_values	- returns the x-axis and/or y-axis values.
fsdata/isfield	- tests if the given field is one of the object properties.
fsdata/ne	- overloads the ~= operator for fsdata objects.
fsdata/reshapeF	- reshape the frequency vector to match the xx vector.
fsdata/save	- an fsdata object to file.
fsdata/set	- set a fsdata property.
fsdata/setFreq	- sets the frequency vector of a fsdata object based on the length of the xx vector and the sample rate.
fsdata/set_xy_axis	- set the x-axis and y-axis values.
fsdata/single_operation	- implements specified operator overload for fsdata objects.
fsdata/subsasgn	- define index assignment for fsdata properties.
fsdata/subsref	- Define field name indexing for fsdata objects.

fsdata/xml	- convert an fsdata object to an DOM XML object.
fsdata/xmladd	- add a fsdata object to the given node of the input XML DOM
fsdata/xmlparse	- creates a fsdata from a fsdata xml file.

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@fsdata](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create a fsdata object.

SYNOPSIS

function data = fromxml(xml)

DESCRIPTION

FROMXML read from part of an XML tree and create a fsdata object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.1 2007/08/29 18:44:38 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE fsdata objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function data = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create a fsdata object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.1 2007/08/29 18:44:38 hewitson Exp $
```

```

0008 %
0009
0010 % defaults
0011 xunits = '';
0012 yunits = '';
0013 dtype = '';
0014
0015 sch = xml.getChildNodes;
0016 for c=1:sch.getLength
0017     x = sch.item(c-1);
0018     switch char(x.getNodeName)
0019         case 'name'
0020             name = char(x.getTextContent);
0021         case 'xunits'
0022             xunits = char(x.getTextContent);
0023         case 'yunits'
0024             yunits = char(x.getTextContent);
0025         case 'version'
0026             version = char(x.getTextContent);
0027         case 'fs'
0028             fs = str2num(char(x.getTextContent));
0029         case 'enbw'
0030             enbw = str2num(char(x.getTextContent));
0031         case 'navs'
0032             navs = str2num(char(x.getTextContent));
0033         case 'created'
0034             xch = x.getChildNodes;
0035             created = time(xch.item(1));
0036         case 'shape'
0037             shape = str2num(char(x.getTextContent));
0038         case 'dtype'
0039             dtype = char(x.getTextContent);
0040         case 'data'
0041             idata = ltpda_xml_read_datasamples(x, dtype);
0042         otherwise
0043     end
0044 end
0045
0046 % build fsdata
0047 data = fsdata();
0048 data = set_xy_axis (data, idata.x, idata.y);
0049 data = set(data, 'enbw', enbw);
0050 data = set(data, 'navs', navs);
0051 data = set(data, 'fs', fs);
0052

```

```
0053 data = set(data, 'name', name);  
0054 data = set(data, 'version', version);  
0055 data = set(data, 'created', created);  
0056 data = set(data, 'xunits', xunits);  
0057 data = set(data, 'yunits', yunits);  
0058  
0059  
0060 % END
```

Generated on Tue 22-Jan-2008 10:39:13 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

```
0001 function ltpda_zaxis(x1,x2)
0002 % LTPDA_ZAXIS Set the Z axis range of the current figure
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_ZAXIS Set the Z axis range of the current figure
0007 %
0008 % CALL:            ltpda_zaxis(x1,x2)
0009 %
```

```
0010 % VERSION:      $Id: ltpda_zaxis.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
0011 %
0012 % HISTORY:       09-10-2001 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 set(gca, 'ZLim', [x1 x2]);
0018
0019 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

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.

%%%

THIS STILL NEEDS FINISHED
THIS STILL NEEDS FINISHED
THIS STILL NEEDS FINISHED
THIS STILL NEEDS FINISHED
THIS STILL NEEDS FINISHED

DESCRIPTION: SINGLE_OPERATION implements specified operator overload
for xyzdata objects.
ATTENTION: Should only be called from a analysis function.

CALL: [h, data_out] = single_operation(a.data, 'log', pl)

varargin format: 1: [xyzdata]
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>
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.
```

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

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SUBFUNCTIONS

- `function pl_default = getDefaultPL()`

SOURCE CODE

```
0001 function varargout = single_operation (varargin)
0002 % SINGLE_OPERATION implements specified operator overload for xyzdata objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % THIS STILL NEEDS FINISHED
0007 % THIS STILL NEEDS FINISHED
0008 % THIS STILL NEEDS FINISHED
0009 % THIS STILL NEEDS FINISHED
0010 % THIS STILL NEEDS FINISHED
0011 %
0012 % DESCRIPTION: SINGLE_OPERATION implements specified operator overload
0013 %               for xyzdata objects.
0014 %               ATTENTION: Should only be called from a analysis function.
0015 %
0016 % CALL: [h, data_out] = single_operation(a.data, 'log', pl)
0017 %
0018 % varargin format: 1:   [xyzdata]
0019 %                  2:   ['operation']
0020 %                  3:   [plist]
0021 %
0022 % varargout format 1: [new_history]
0023 %                  2: [data_out1]
```

```

0024 %          3: [data_out2]          optional
0025 %          4: [data_out3]          optional
0026 %          ...                    optional
0027 %
0028 % With the plist values you can control the command line.
0029 % Possible key values are: option   <double> or <char>
0030 %                          dim      <double> or <char>
0031 %                          W        <vector>
0032 %                          xdata    <char>
0033 %                          ydata    <char>
0034 %                          xdata    <char>
0035 % With the keys xdata,ydata or zdata you can control which data are to compute.
0036 % Attention the value of the key 'xdata', 'ydata' or 'zdata' must apply to the
0037 % data class.
0038 % The other values are to control the command line:
0039 %
0040 % >> operation(data,'option')      option <char>
0041 % >> operation(data,option)         option <double>
0042 % >> operation(data,W,dim)
0043 % >> operation(data,dim)
0044 % >> operation(data,W)
0045 % >> operation(data)
0046 %
0047 % HISTORY: 24-12-2007 Diepholz
0048 %          Creation.
0049 %
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051
0052 VERSION = '$Id: single_operation.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $';
0053
0054 % Check if this is a call for parameters
0055 if nargin == 2
0056     if ischar(varargin{2})
0057         in = char(varargin{2});
0058         if strcmp(in, 'Params')
0059             varargout{1} = getDefaultPL();
0060             return
0061         elseif strcmp(in, 'Version')
0062             varargout{1} = VERSION;
0063             return
0064         end
0065     end
0066 end
0067
0068 % Only the call from an analysis object is allowed.
0069 if (nargin == 3)
0070
0071     data = varargin {1};
0072     op   = varargin {2};
0073     pl   = varargin {3};
0074     a    = varargin {end};
0075     x    = data.x;
0076     y    = data.y;

```



```

0077     z      = data.z;
0078
0079 %% Verify input variables and set default parameter
0080 if ~isa(data, 'xyzdata')
0081     error('### xyzdata/single_operation: The first input must be a xyzdata.');
```

```

0082 end
0083 if ~ischar(op)
0084     error('### xyzdata/single_operation: The second input must be a string.');
```

```

0085 end
0086 if ~isa(plist, 'plist')
0087     error('### xyzdata/single_operation: The third input must be a plist.');
```

```

0088 end
0089
0090 %% Set default parameter list
0091 pl_default = getDefaultPL();
0092
0093 %% Combine the handover param-list and the default param-list
0094 if ~isempty(pl)
0095     pl = combine(pl);
0096 else
0097     pl = combine(pl_default);
0098 end
0099
0100 option      = find(pl, 'option');
0101 dim         = find(pl, 'dim');
0102 weight_vector = find(pl, 'w');
0103 do_xdata    = find(pl, 'xdata');
0104 do_ydata    = find(pl, 'ydata');
0105 do_zdata    = find(pl, 'zdata');
```

```

0106
0107 %% Check the <char> option if it begins and ends with a quote '
0108 %% If not then add the quote.
0109 if ischar(option) && ~isempty(option)
0110     if ~strcmp(option(1), '')
0111         option = ['' option];
0112     end
0113     if ~strcmp(option(end), '')
0114         option = [option ''];
0115     end
0116 end
0117
0118 %% Convert the <double> option into a char
0119 if isnumeric(option) || islogical(option)
0120     option = num2str(option);
0121 end
0122
0123 %% Convert the dim into a char
0124 if isnumeric(dim) || islogical(dim)
0125     dim = num2str(dim);
0126 end
0127
0128 %% The operation can only compute with a 'option' or with a 'dimension'.
0129 %% e.g. operation(ao, 'option')
```

```

0130 %%      operation(ao, dim)
0131 if ~isempty(option) && ~isempty(dim)
0132     warning ('### xyzdata/single_operation: The param list contain a ''option''
and a ''dim''. It is computed %s(ao,option).', op);
0133 end
0134
0135 %%
0136 if isempty(do_xdata) && isempty(do_ydata)
0137     do_ydata = 'y';
0138 end
0139
0140 %% %% Ensure that the called function have at least two output variables
0141 if (nargout >= 2)
0142
0143 %% %% xdata:
0144 %% Create the output string for the eval command
0145 %% e.g.: x_new = '[ x_new1 x_new2 ]'
0146 x_new = '[';
0147 for i = 1:nargout-1
0148     new_var_name = genvarname (sprintf('x_new%d',i));
0149     eval ([new_var_name '= 0;']);
0150
0151     x_new = strcat(x_new, sprintf (' x_new%d',i));
0152 end
0153 x_new = strcat (x_new, ' ');
0154
0155 %% ydata:
0156 %% Create the output string for the eval command
0157 %% e.g.: y_new = '[ y_new1 y_new2 ]'
0158 y_new = '[';
0159 for i = 1:nargout-1
0160     new_var_name = genvarname (sprintf('y_new%d',i));
0161     eval ([new_var_name '= 0;']);
0162
0163     y_new = strcat(y_new, sprintf (' y_new%d',i));
0164 end
0165 y_new = strcat (y_new, ' ');
0166
0167 %% %% Create the output variables
0168 %% e.g.: data_out1 = data;
0169 for i = 1:nargout-1
0170     new_var_name = genvarname (sprintf('data_out%d',i));
0171     eval ([new_var_name '= data;']);
0172 end
0173
0174 %% Execute the command for the xdata
0175 if ~isempty(do_xdata)
0176
0177     try
0178         %% Execute with a option
0179         %% >> operation(x,'option');
0180         if ~isempty(option)
0181             cmd_line = sprintf('%s = %s(x,%s);',x_new, op, option);

```

```

0182         eval (cmd_line);
0183
0184         %% Execute with a weight vector and dimension
0185         %% >> operation(x,W,dim);
0186         elseif ~isempty(dim) && ~isempty(weight_vector)
0187             cmd_line = sprintf('%s = %s(x,weight_vector,%s);',x_new, op, dim);
0188             eval (cmd_line);
0189
0190         %% Execute with a dimension
0191         %% >> operation(x,dim);
0192         elseif ~isempty(dim)
0193             cmd_line = sprintf('%s = %s(x,%s);',x_new, op, dim);
0194             eval (cmd_line);
0195
0196         %% Execute with a weight vector
0197         %% >> operation(x,W);
0198         elseif ~isempty(weight_vector)
0199             cmd_line = sprintf('%s = %s(x,weight_vector);',x_new, op);
0200             eval (cmd_line);
0201
0202         %% Execute without parameter or option
0203         %% >> operation(x)
0204         else
0205             cmd_line = sprintf('%s = %s(x);',x_new, op);
0206             eval (cmd_line);
0207         end
0208
0209     catch
0210         l_error = lasterror;
0211         disp(sprintf('\n\n%s', l_error.message));
0212         error('### cdata/single_operation: Can not command:\n###      %s\n###
Have a look to the param list', cmd_line);
0213     end
0214
0215 end
0216
0217 %% Execute the command for the ydata
0218 if ~isempty(do_ydata)
0219     try
0220         %% Execute with a option or a dimension
0221         %% >> operation(y,'option');
0222         if ~isempty(option)
0223             cmd_line = sprintf('%s = %s(y,%s);',y_new, op, option);
0224             eval (cmd_line);
0225
0226         %% Execute with a weight vector and dimension
0227         %% >> operation(y,W,dim);
0228         elseif ~isempty(dim) && ~isempty(weight_vector)
0229             cmd_line = sprintf('%s = %s(y,weight_vector,%s);',y_new, op, dim);
0230             eval (cmd_line);
0231
0232         %% Execute with a option or a dimension
0233

```

```

0234     %% >> operation(y,dim);
0235 elseif ~isempty(dim)
0236     cmd_line = sprintf('%s = %s(y,%s);',y_new, op, dim);
0237     eval (cmd_line);
0238
0239     %% Execute with a weight vector
0240     %% >> operation(y,W);
0241 elseif ~isempty(weight_vector)
0242     cmd_line = sprintf('%s = %s(y,weight_vector);',y_new, op);
0243     eval (cmd_line);
0244
0245     %% Execute without parameter or option
0246     %% >> operation(y)
0247 else
0248     cmd_line = sprintf('%s = %s(y);',y_new, op);
0249     eval (cmd_line);
0250 end
0251
0252 catch
0253     l_error = lasterror;
0254     disp(sprintf('\n\n%s', l_error.message));
0255     error('### cdata/single_operation: Can not command:\n###      %s\n###
Have a look to the param list', cmd_line);
0256 end
0257
0258 end
0259
0260 %% %% The result from eval is in x_new1, y_new1, ...
0261 %% Create a xydata object from this results and write it into the output
0262 for i = 1:nargout-1
0263
0264     %% Create xydata
0265     if ~isempty(do_xdata)
0266         eval (sprintf('data_out%d.x = x_new%d;',i,i));
0267         eval (sprintf('data_out%d.name = ''%s(%s)'';',i,op,data.name));
0268         eval (sprintf('data_out%d.xunits = ''%s(%s)'';',i,op,data.xunits));
0269     end
0270     if ~isempty(do_ydata)
0271         eval (sprintf('data_out%d.y = y_new%d;',i,i));
0272         eval (sprintf('data_out%d.name = ''%s(%s)'';',i,op,data.name));
0273         eval (sprintf('data_out%d.yunits = ''%s(%s)'';',i,op,data.yunits));
0274     end
0275
0276     %% Write to output
0277     eval (sprintf('varargout{i+1} = data_out%d;',i))
0278
0279 end
0280
0281 %% Create a new history object
0282 h = history(op, VERSION);
0283
0284 varargout{1} = h;
0285

```

```
0286     else % (nargout < 2)
0287         error ('### xydata/single_operation: This function need at least two output
variables. ');
0288     end
0289
0290 else
0291     error ('### xydata/single_operation: This function need three input
variables. ')
0292 end
0293
0294 function pl_default = getDefaultPL()
0295
0296     pl_default = plist('option', '', ...
0297                       'dim',    '', ...
0298                       'w',      [], ...
0299                       'xdata',  '', ...
0300                       'ydata',  'y');
0301
```

[Home](#) > [classes](#) > [@fsdata](#) > [private](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@history](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create a history object.

SYNOPSIS

```
function h = fromxml(xml)
```

DESCRIPTION

FROMXML read from part of an XML tree and create a history object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.1 2007/08/29 18:44:11 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE history objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function h = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create a history object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.1 2007/08/29 18:44:11 hewitson Exp $
0008 %
```

```

0009
0010 h = history;
0011
0012 sch = xml.getChildNodes;
0013 for c=1:sch.getLength
0014     x = sch.item(c-1);
0015     switch char(x.getNodeName)
0016         case 'name'
0017             h = set(h, 'name', char(x.getTextContent));
0018         case 'version'
0019             h = set(h, 'version', char(x.getTextContent));
0020         case 'created'
0021             xch = x.getChildNodes;
0022             h = set(h, 'created', time(xch.item(1)));
0023         case 'n'
0024             h = set(h, 'n', str2num(char(x.getTextContent)));
0025         case 'pn'
0026             h = set(h, 'pn', str2num(char(x.getTextContent)));
0027         case 'consver'
0028             h = set(h, 'consver', char(x.getTextContent));
0029         case 'invars'
0030             h = set(h, 'invars', ltpda_str2cells(char(x.getTextContent)));
0031         case 'plist'
0032             xch = x.getChildNodes;
0033             h = set(h, 'plist', plist(xch.item(1)));
0034         case 'inhists'
0035             xch = x.getChildNodes;
0036             hs = [];
0037             for j=1:xch.getLength
0038                 hch = xch.item(j-1);
0039                 if strcmp(char(hch.getNodeName), 'history')
0040                     hs = [hs history(hch)];
0041                 end
0042             end
0043             h = set(h, 'inhists', hs);
0044         otherwise
0045     end
0046 end
0047
0048 % END

```


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

```
0001 function ltpda_zscale(scale)
0002 % LTPDA_ZSCALE Set the Z scale of the current axis
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_ZSCALE Set the Z scale of the current axis
0007 %
0008 % CALL:            ltpda_zscale('scale')    scale = 'lin' or 'log';
0009 %
```

```
0010 % VERSION:      $Id: ltpda_zscale.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
0011 %
0012 % HISTORY:       02-05-2004 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 set(gca, 'ZScale', scale);
0018
0019 % END
```

Generated on Tue 22-Jan-2008 10:39:13 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.

history/xmladd	- add an history
object to the given node of the input XML DOM	
history/xmlparse	- creates a
history from a history xml file.	

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@mfir](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create an mfir object.

SYNOPSIS

function fout = fromxml(xml)

DESCRIPTION

FROMXML read from part of an XML tree and create an mfir object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.3 2007/11/02 13:18:29 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE mfir objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function fout = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create an mfir object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.3 2007/11/02 13:18:29 ingo Exp $
```

```

0008 %
0009
0010 % disp('----- miir -----')
0011 % xml.getNodeName
0012 % disp('----- miir -----')
0013
0014 sch = xml.getChildNodes;
0015 for c=1:sch.getLength
0016     x = sch.item(c-1);
0017     switch char(x.getNodeName)
0018         case 'name'
0019             f.name = char(x.getTextContent);
0020         case 'fs'
0021             f.fs = str2num(char(x.getTextContent));
0022         case 'ntaps'
0023             f.ntaps = str2num(char(x.getTextContent));
0024         case 'a'
0025             f.a = str2num(char(x.getTextContent));
0026         case 'gd'
0027             f.gd = str2num(char(x.getTextContent));
0028         case 'gain'
0029             f.gain = str2num(char(x.getTextContent));
0030         case 'histout'
0031             f.histout = str2num(char(x.getTextContent));
0032         case 'infile'
0033             f.infile = char(x.getTextContent);
0034         case 'plist'
0035             xch = x.getChildNodes;
0036             f.plist = plist(xch.item(1));
0037         case 'created'
0038             xch = x.getChildNodes;
0039             f.created = time(xch.item(1));
0040         case 'version'
0041             f.version = char(x.getTextContent);
0042     otherwise
0043     end
0044 end
0045
0046 % build filter
0047 fout = mfir(f);
0048
0049 % END

```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@history](#) > [private](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@miir](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create an miir object.

SYNOPSIS

```
function fout = fromxml(xml)
```

DESCRIPTION

FROMXML read from part of an XML tree and create an miir object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.2 2007/10/24 10:55:11 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE miir objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function fout = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create an miir object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.2 2007/10/24 10:55:11 ingo Exp $
```

```

0008 %
0009
0010 % disp('----- miir -----')
0011 % xml.getNodeName
0012 % disp('----- miir -----')
0013
0014 sch = xml.getChildNodes;
0015 for c=1:sch.getLength
0016     x = sch.item(c-1);
0017     switch char(x.getNodeName)
0018         case 'version'
0019             f.version = char(x.getTextContent);
0020         case 'name'
0021             f.name = char(x.getTextContent);
0022         case 'created'
0023             xch = x.getChildNodes;
0024             f.created = time(xch.item(1));
0025         case 'fs'
0026             f.fs = str2num(char(x.getTextContent));
0027         case 'ntaps'
0028             f.ntaps = str2num(char(x.getTextContent));
0029         case 'gain'
0030             f.gain = str2num(char(x.getTextContent));
0031         case 'infile'
0032             f.infile = char(x.getTextContent);
0033         case 'plist'
0034             xch = x.getChildNodes;
0035             f.plist = plist(xch.item(1));
0036         case 'a'
0037             f.a = str2num(char(x.getTextContent));
0038         case 'b'
0039             f.b = str2num(char(x.getTextContent));
0040         case 'histin'
0041             f.histin = str2num(char(x.getTextContent));
0042         case 'histout'
0043             f.histout = str2num(char(x.getTextContent));
0044         otherwise
0045     end
0046 end
0047
0048 % build filter
0049 fout = miir(f);
0050
0051 % END

```

Generated on Tue 22-Jan-2008 10:39:13 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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@param](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create a param object.

SYNOPSIS

```
function p = fromxml(xml)
```

DESCRIPTION

FROMXML read from part of an XML tree and create a param object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.5 2007/10/26 14:38:15 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE param objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function p = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create a param object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.5 2007/10/26 14:38:15 hewitson Exp $
```

```

0008 %
0009
0010 % disp('----- param -----')
0011 % xml.getNodeName
0012 % disp('----- param -----')
0013
0014 p = param();
0015 sch = xml.getChildNodes;
0016 for c=1:sch.getLength
0017     x = sch.item(c-1);
0018     switch char(x.getNodeName)
0019         case 'name'
0020             p.name = char(x.getTextContent);
0021         case 'created'
0022             xch = x.getChildNodes;
0023             p = set(p, 'created', time(xch.item(1)));
0024         case 'version'
0025             p = set(p, 'version', char(x.getTextContent));
0026         case 'key'
0027             p = set(p, 'key', char(x.getTextContent));
0028         case 'value'
0029             % if this has children then there is probably an object
0030             % below this
0031             mych = x.getChildNodes;
0032             if mych.getLength > 1
0033                 % get the object
0034                 vobj = x.item(1);
0035                 switch char(vobj.getNodeName)
0036                     case 'specwin'
0037                         p = set(p, 'val', specwin(vobj));
0038                     case 'miir'
0039                         p = set(p, 'val', miir(vobj));
0040                     case 'mfir'
0041                         p = set(p, 'val', mfir(vobj));
0042                     case 'pole'
0043                         p = set(p, 'val', pole(vobj));
0044                     case 'zero'
0045                         p = set(p, 'val', zero(vobj));
0046                     case 'pzmodel'
0047                         p = set(p, 'val', pzmodel(vobj));
0048                     case 'time'
0049                         p = set(p, 'val', time(vobj));
0050                     case 'timespan'
0051                         p = set(p, 'val', timespan(vobj));

```

```
0052         otherwise
0053             error('### Unknown object in parameter value.');
```

end

```
0055 else
0056     % we have numeric or char parameter
0057     valstr = char(x.getTextContent);
0058     if ~isempty(str2num(valstr))
0059         p = set(p, 'val', str2num(valstr));
0060     else
0061         if length(valstr) >= 5 && strcmp(valstr(1:5), 'CELL:')
0062             p = set(p, 'val', ltpda_str2cells(valstr(6:end)));
0063         else
0064             p = set(p, 'val', valstr);
0065         end
0066     end
0067 end
0068 otherwise
0069 end
0070 end
0071
0072 % build plist
0073
0074 % END
```


[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [sqlResultsGUI.m](#)

sqlResultsGUI

PURPOSE [^](#)

SQLRESULTSGUI shows the results of a query in a new window.

SYNOPSIS [^](#)

```
function sqlResultsGUI(results, fields, query)
```

DESCRIPTION [^](#)

SQLRESULTSGUI shows the results of a query in a new window.

```
sqlResultsGUI(results, fields)
```

M Hewitson 09-07-07

```
$Id: sqlResultsGUI.m,v 1.3 2007/11/13 17:20:55 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^](#)

This function calls:

- [mutable](#) UITABLE creates a two dimensional graphic uitable component in a figure window.

This function is called by:

SUBFUNCTIONS [^](#)

- [function exportBtn_Callback\(hObject, eventdata, gui, results, fields\)](#)
- [function resizeMain\(hObject, eventdata, leftMargin, bottomMargin, topMargin\)](#)

SOURCE CODE [^](#)

```
0001 function sqlResultsGUI(results, fields, query)
0002
0003 % SQLRESULTSGUI shows the results of a query in a new window.
0004 %
0005 % sqlResultsGUI(results, fields)
0006 %
```

```

0007 % M Hewitson 09-07-07
0008 %
0009 % $Id: sqlResultsGUI.m,v 1.3 2007/11/13 17:20:55 hewitson Exp $
0010 %
0011
0012 leftMargin    = 10;
0013 bottomMargin  = 100;
0014 topMargin     = 20;
0015
0016 btnHeight     = 20;
0017 btnWidth      = 60;
0018 btnGap        = 20;
0019
0020 Color         = [240 240 240]/255;
0021 Height        = 600;
0022 Width         = 800;
0023
0024 Screen        = get(0, 'screensize');
0025 Position      = [Screen(3)/2 - Width/2 Screen(4)/2-Height/2 Width Height];
0026
0027 % create figure
0028 gui.main = figure('HandleVisibility','on',...
0029     'IntegerHandle','off',...
0030     'Menubar','none',...
0031     'NumberTitle','off',...
0032     'Name','Query Results',...
0033     'Tag','main',...
0034     'Color',Color,...
0035     'Resize','on',...
0036     'Position',Position,...
0037     'ResizeFcn',{@resizeMain, leftMargin, bottomMargin, topMargin});
0038
0039 % add query string
0040 Tleft    = leftMargin;
0041 Twidth   = Width - 2*leftMargin;
0042 Theight  = 100;
0043 Tbottom  = Position(4) - Theight - 20;
0044 sth = uicontrol(gui.main, 'Style','text',...
0045     'String',query,...
0046     'FontSize', 14,...
0047     'BackgroundColor',Color,...
0048     'HorizontalAlignment','left',...
0049     'Position',[Tleft Tbottom Twidth Theight]);
0050
0051
0052 % add table
0053 Tleft    = leftMargin;
0054 Twidth   = Width - 2*leftMargin;
0055 Theight  = 400;
0056 Tbottom  = 100;

```

```

0057 t = muitable(gui.main, results, fields, 'Position', ...
0058     [Tleft Tbottom Twidth Theight]);
0059 set(t, 'Editable', 0);
0060
0061 setappdata(gui.main, 'table', t);
0062
0063
0064 % add export button
0065 btn = uicontrol(gui.main, ...
0066     'Style', 'pushbutton', ...
0067     'String', 'Export', ...
0068     'Tag', 'exportBtn', ...
0069     'Callback', {@exportBtn_Callback, gui, results, fields}, ...
0070     'Position', [leftMargin leftMargin btnWidth btnHeight]);
0071
0072
0073
0074
0075 %-----
0076 % Callbacks
0077
0078 %-----
0079 % export button
0080 function exportBtn_Callback(hObject, eventdata, gui, results, fields)
0081
0082 % get table
0083 t = getappdata(gui.main, 'table');
0084 table = get(t.Table);
0085
0086 rows = table.SelectedRows + 1;
0087 cols = table.SelectedColumns + 1;
0088 res = results(rows, cols);
0089 nResults = size(res, 1);
0090
0091 % get file to export to
0092 [filename, pathname, filteridx] = uiputfile('*.txt', 'Select export file');
0093
0094 fd = fopen(filename, 'w+');
0095
0096 % header
0097 fprintf(fd, '# ');
0098 fprintf(fd, '%s\t|\t', fields{cols});
0099 fprintf(fd, '\n');
0100
0101
0102 for j=1:nResults
0103     for k=1:size(res,2)
0104         r = res{j,k};
0105         if ischar(r)
0106             fprintf(fd, '%s\t\t', char(r));

```

```
0107     elseif isnumeric(r)
0108         fprintf(fd, '%f\t\t', r);
0109     else
0110         error('### Unknown data format. ');
0111     end
0112 end
0113 fprintf(fd, '\n');
0114 end
0115 fclose(fd);
0116
0117 edit(filename);
0118
0119 %-----
0120 % Resize Main figure
0121 function resizeMain(hObject, eventdata, leftMargin, bottomMargin, topMargin)
0122
0123 % resize table
0124 t = getappdata(hObject, 'table');
0125 fs = get(hObject, 'Position');
0126 pos = get(t, 'Position');
0127
0128 set(t, 'Position', [pos(1) pos(2) fs(3)-2*leftMargin fs(4)-bottomMargin-
topMargin]);
0129
0130
0131 % END
```

[Home](#) > [classes](#) > [@mfir](#) > [private](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@plist](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create a plist object.

SYNOPSIS

```
function plo = fromxml(xml)
```

DESCRIPTION

FROMXML read from part of an XML tree and create a plist object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.2 2007/10/15 17:16:56 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE plist objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function plo = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create a plist object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.2 2007/10/15 17:16:56 ingo Exp $
```

```

0008 %
0009
0010 % disp('----- plist -----')
0011 % xml.getNodeName
0012 % disp('----- plist -----')
0013
0014 plo = plist();
0015
0016 plo.params = [];
0017 sch = xml.getChildNodes;
0018 for c=1:sch.getLength
0019     x = sch.item(c-1);
0020     switch char(x.getNodeName)
0021         case 'name'
0022             plo.name = char(x.getTextContent);
0023         case 'version'
0024             plo.version = char(x.getTextContent);
0025         case 'created'
0026             xch = x.getChildNodes;
0027             plo.created = time(xch.item(1));
0028         case 'params'
0029             % read each param
0030             ps = x.getChildNodes;
0031             for j=1:ps.getLength
0032                 pn = ps.item(j-1);
0033                 if strcmp((pn.getNodeName), 'param')
0034                     plo.params = [plo.params param(pn)];
0035                 end
0036             end
0037         otherwise
0038     end
0039 end
0040
0041
0042 % END

```

[Home](#) > [classes](#) > [@miir](#) > [private](#) > mfilload.m

mfilload

PURPOSE

MFILTLOAD Load in a filter from *.filt files.

SYNOPSIS

function f = mfilload(filename)

DESCRIPTION

MFILTLOAD Load in a filter from *.filt files.

usage: f = mfilload(filename)

M Hewitson 11-02-07

\$Id: mfilload.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

```
0001 function f = mfilload(filename)
0002
0003 % MFILTLOAD Load in a filter from *.filt files.
0004 %
0005 % usage: f = mfilload(filename)
0006 %
0007 % M Hewitson 11-02-07
0008 %
0009 %
0010 % $Id: mfilload.m,v 1.2 2007/10/24 10:55:11 ingo Exp $
0011 %
0012
0013
0014 delims = ' \n';
```



```

0015 [in, msg] = fopen(filename, 'rt');
0016
0017 if(in < 0)
0018     error(['can not open file ' filename]);
0019 end
0020
0021
0022 n = 1;
0023 ncoeffs = 1;
0024 filt.a = [];
0025 filt.b = [];
0026 norminst = '';
0027
0028 lbuffer = fgets(in);
0029
0030 while (lbuffer > 0)
0031     if(length(lbuffer) > 1)
0032         % read the first token on this line
0033         [token, r] = strtok (lbuffer, delims);
0034         % ignore comments
0035         if(token(1) ~= '#')
0036             if(strcmp(token, 'NAME')) % name tag
0037                 % get next token as name
0038                 [token, r] = strtok (r, delims);
0039                 if(length(token) > 0)
0040                     filt.name = token;
0041                 end
0042             elseif(strcmp(token, 'GAIN'))
0043                 % get next token as gain
0044                 [token, r] = strtok (r, delims);
0045                 if(length(token) > 0)
0046                     filt.gain = str2num(token);
0047                     if(filt.gain < 0 | filt.gain > 1e20)
0048                         error(['### gain is a strange number (' num2str(filt.gain) ')']);
0049                     end
0050                 end
0051             elseif(strcmp(token, 'FS'))
0052                 % get next token as fs
0053                 [token, r] = strtok (r, delims);
0054                 if(length(token) > 0)
0055                     filt.fs = str2num(token);
0056                     if(filt.fs < 0 | filt.fs > 1e10)
0057                         error(['### sample rate is a strange number (' num2str(filt.fs)
0058                             ')']);
0059                     end
0060                 end
0061             elseif(strcmp(token, 'NTAPS'))
0062                 % get next token as ntaps
0063                 [token, r] = strtok (r, delims);
0064                 if(length(token) > 0)
0065                     filt.ntaps = str2num(token);

```

```

0065         if(filt.ntaps <= 0 | filt.ntaps > 20)
0066             error('### number of taps is out of bounds (1->10)');
0067         end
0068     end
0069 elseif(strcmp(token, 'NORM'))
0070     % get next token as normalisation instruction
0071     [token, r] = strtok (r, delims);
0072     norminst = token;
0073 elseif(strcmp(token, 'COEFF'))
0074     if(ncoeffs <= filt.ntaps)
0075         % get next token as A coeff
0076         [token, r] = strtok (r, delims);
0077         if(length(token) == 0)
0078             filt.a(ncoeffs) = 0;
0079         else
0080             filt.a(ncoeffs) = str2num(token);
0081         end
0082
0083         % get next token as B coeff
0084         [token, r] = strtok (r, delims);
0085         if(length(token) == 0)
0086             filt.b(ncoeffs) = 0;
0087         else
0088             filt.b(ncoeffs) = str2num(token);
0089         end
0090
0091         ncoeffs = ncoeffs + 1;
0092     else
0093         error(['### max taps reached. Ignoring line ' num2str(n) ]);
0094     end
0095 else
0096     warning(['### keyword not found (line ' num2str(n) ') ']);
0097 end
0098 end
0099 end
0100 n = n + 1;
0101 lbuffer = fgets(in);
0102 end
0103
0104 % close file
0105 fclose(in);
0106
0107 % before returning, normalise gain if required
0108 if(length(norminst) > 0)
0109     if(strncmp(norminst, 'dc', 2))
0110         % normalise dc to unity
0111         gain = iirdcgain(filt.a, filt.b);
0112         filt.a = filt.a ./ gain;
0113     elseif(strncmp(norminst, 'hf', 2))
0114         % normalise hf to unity
0115         gain = iirhfgain(filt.a, filt.b);

```

```
0116      filt.a = filt.a ./ gain;
0117      else
0118          error(['### readfilt: normalisation instruction (' norminst ') not
recognised.']);
0119      end
0120 end
0121
0122 f.name      = filt.name;
0123 f.fs        = filt.fs;
0124 f.ntaps     = filt.ntaps;
0125 f.a         = filt.a;
0126 f.b         = filt.b;
0127 f.gain      = filt.gain;
0128 f.histin    = zeros(1, f.ntaps-1);
0129 f.histout   = zeros(1, f.ntaps-1);
0130
0131
0132 % END
```

[Home](#) > [classes](#) > [@miir](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@pole](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create a zero object.

SYNOPSIS

```
function po = fromxml(xml)
```

DESCRIPTION

FROMXML read from part of an XML tree and create a zero object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.2 2007/10/16 17:37:21 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE pole objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function po = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create a zero object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.2 2007/10/16 17:37:21 ingo Exp $
```

```

0008 %
0009
0010
0011 % disp('----- pole -----')
0012 % xml.getNodeName
0013 % disp('----- pole -----')
0014
0015 sch = xml.getChildNodes;
0016 for c=1:sch.getLength
0017     x = sch.item(c-1);
0018     switch char(x.getNodeName)
0019         case 'version'
0020             p.version = char(x.getTextContent);
0021         case 'name'
0022             p.name = char(x.getTextContent);
0023         case 'f'
0024             p.f = str2num(char(x.getTextContent));
0025         case 'q'
0026             p.q = str2num(char(x.getTextContent));
0027         case 'created'
0028             xch = x.getChildNodes;
0029             p.created = time(xch.item(1));
0030         otherwise
0031     end
0032 end
0033
0034 % build pole
0035 po = pole(p.f, p.q);
0036 po = set(po, 'name', p.name);
0037 po = set(po, 'version', p.version);
0038 po = set(po, 'created', p.created);
0039
0040 % END

```

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 miir object.
f = mfir(fi)    - creates a copy of the input miir object, fi.
f = mfir(a)      - creates a an miir object based on the magnitude of the
                  input A0/fsdata object.
f = mfir(pl)     - create an miir object from the description given in the
                  parameter list.

```

Parameters for filter design from A0/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 standard filters:

```

'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 0.3] ]
'Win'      - Specify window function used in filter design [default: Hamming]

```

Example 1:

Create a default lowpass filter.

```
>> f = mfir(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('gain', 2.0))
>> pl = append(pl, param('fs', 1000))
>> pl = append(pl, param('fc', 200))
>> f = mfir(pl)
```

M Hewitson 09-02-07

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.
- [xmlparse](#) XMLPARSE creates a mfir filter from a mfir xml file.

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.

SUBFUNCTIONS

- [function filt = mfirFromAO\(ALGONAME, VERSION, a, pl\)](#)
- [function pl = getDefaultPlist\(\)](#)
- [function f = init\(VERSION\)](#)

SOURCE CODE

```
0001 function f = mfir(varargin)
0002 % MFIR FIR filter object class constructor.
0003 %     Create an mfir object.
0004 %
0005 %     f = mfir()      - creates an empty miir object.
0006 %     f = mfir(fi)   - creates a copy of the input miir object, fi.
0007 %     f = mfir(a)    - creates a an miir object based on the magnitude of the
0008 %                     input A0/fsdata object.
0009 %     f = mfir(pl)   - create an miir object from the description given in the
0010 %                     parameter list.
0011 %
0012 % Parameters for filter design from A0/fsdata:
0013 %     'method' - design method:
0014 %               'frequency-sampling' - uses fir2()
```

```

0015 %          'least-squares'      - uses firls()
0016 %          'Parks-McClellan'    - uses firpm()
0017 %      'Win'      - Window function for frequency-sampling method
0018 %      'N'        - filter order
0019 %
0020 % Parameters for standard filters:
0021 %
0022 %      'type'      - one of 'highpass', 'lowpass', 'bandpass', 'bandreject'.
0023 %                  [default: 'lowpass']
0024 %      'gain'      - gain of filter [default: 1.0]
0025 %      'fs'        - sample frequency to design for [default: 1Hz]
0026 %      'order'     - order of filter [default: 1]
0027 %      'fc'        - corner frequencies. This is a two element vector for bandpass
0028 %                  and bandreject filters. [default: 0.25 or [0.1 0.3] ]
0029 %      'Win'       - Specify window function used in filter design [default: Hamming]
0030 %
0031 % Example 1:
0032 %
0033 % Create a default lowpass filter.
0034 %     >> f = mfir(plist())
0035 %
0036 % Example 2:
0037 %
0038 % Create an order 1 highpass filter with high frequency gain 2. Filter is
0039 % designed for 1kHz sampled data and has a cut-off frequency of 200Hz.
0040 %     >> pl = plist();
0041 %     >> pl = append(pl, param('type', 'highpass'))
0042 %     >> pl = append(pl, param('gain', 2.0))
0043 %     >> pl = append(pl, param('fs', 1000))
0044 %     >> pl = append(pl, param('fc', 200))
0045 %     >> f = mfir(pl)
0046 %
0047 %
0048 % M Hewitson 09-02-07
0049 %
0050
0051 ALGONAME = mfilename;
0052 VERSION  = '$Id: mfir.m,v 1.14 2007/12/12 18:45:08 ingo Exp $';
0053
0054 %%%%% Is this a 'Params' call? %%%%%
0055 if nargin == 2 && isa(varargin{1}, 'mfir') && ischar(varargin{2})
0056     if strcmp(varargin{2}, 'Params')
0057         f = getDefaultPlist();
0058         return
0059     elseif strcmp(varargin{2}, 'Version')
0060         f = VERSION;
0061         return
0062     end
0063 end
0064
0065 switch nargin
0066
0067     %%%%%%%%% f = mfir() %%%%%%%%%

```

```

0068 % create default mfir object
0069 case 0
0070     f = init(VERSION);
0071
0072 % Copy, standard
0073 case 1
0074
0075 %%%%%%%%% Create from XML fragment %%%%%%%%%
0076 if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0077     f = fromxml(varargin{1});
0078 %%%%%%%%% From File %%%%%%%%%
0079 elseif ischar(varargin{1})
0080
0081     filename = varargin{1};
0082     [path, name, ext, vers] = fileparts(filename);
0083     switch ext
0084     case '.mat'
0085         f = load(filename);
0086         f = f.a;
0087     case '.xml'
0088         f = xmlparse(mfir, filename);
0089     otherwise
0090         error('### Unknown file type.');
```

0091 end

```

0092 %%%%%%%%% f = mfir(mfir) %%%%%%%%%
0093 elseif isa(varargin{1}, 'mfir')
0094     f = varargin{1};
0095     f.version = VERSION;
0096
0097 %%%%%%%%% f = mfir(ao) %%%%%%%%%
0098 elseif isa(varargin{1}, 'ao')
0099     pl = getDefaultPlist();
0100     f = mfirFromAO(ALGONAME, VERSION, varargin{1}, pl);
0101
0102 %%%%%%%%% f = mfir(struct) %%%%%%%%%
0103 elseif isstruct(varargin{1})
0104
0105     f = init(VERSION);
0106
0107     fields = fieldnames(varargin{1});
0108     for ii = 1:length(fields)
0109         field = fields{ii};
0110
0111         %% plist -> plist-object
0112         if strcmp(field, 'plist')
0113             if isstruct(varargin{1}.(field))
0114                 f.(field) = plist(varargin{1}.(field));
0115             else
0116                 f.(field) = varargin{1}.(field);
0117             end
0118         %% created -> time-object
0119         elseif strcmp(field, 'created')
```

```

0120         created = varargin{1}.created;
0121         if isstruct(created)
0122             created = time(created);
0123         end
0124         f.created      = created;
0125         %%% All other
0126         else
0127             try
0128                 f.(field) = varargin{1}.(field);
0129             catch
0130                 error('### The field '%s'' in the struct is not a mfir property.',
field)
0131             end
0132         end
0133     end
0134
0135     %%%%%%%%%%% f = mfir(plist) %%%%%%%%%%%
0136     % Parameter list
0137     elseif isa(varargin{1}, 'plist')
0138
0139         pl  = varargin{1};
0140         type = find(pl, 'type');
0141         pzm  = find(pl, 'pzmodel');
0142
0143         f = init(VERSION);
0144
0145         % deal with empty plist
0146         if isempty(type) && isempty(pzm)
0147             type = 'lowpass';
0148         end
0149
0150         if ~isempty(type)
0151             % check and fill parameter list
0152             plo = parseFilterParams(pl);
0153             switch type
0154                 case 'lowpass'
0155                     f = mklowpass(f, plo);
0156                 case 'highpass'
0157                     f = mkhighpass(f, plo);
0158                 case 'bandpass'
0159                     f = mkbandpass(f, plo);
0160                 case 'bandreject'
0161                     f = mkbandreject(f, plo);
0162                 otherwise
0163                     error('### unknown standard filter type in miir constructor.');
```

```

0172         if isempty(fs)
0173             % get max freq in pzmodel
0174             fs = 8*getupperFreq(pzm);
0175             warning([sprintf('!!! no sample rate specified. Designing for fs=%
2.2fHz.', fs)...
0176                     sprintf('\nThe filter will be redesigned later when
used.')]));
0177
0178         end
0179         % make MIIR filter
0180         f = tomfir(pzm, plist('fs', fs));
0181         f = set(f, 'name', get(pzm, 'name'));
0182         f = set(f, 'plist', pl);
0183
0184     else
0185         error('### unknown constructor type for mfir.');
```

```

0186     end
0187 else
0188     error('### unknown constructor type for mfir.');
```

```

0189 end

0190
0191 case 2 % pzmodel followed by plist
0192
0193     % From spectrum
0194     %%%%%%%%%%% From DATABASE
0195     if isa(varargin{1}, 'database')
0196         f = retrieve(varargin{1}, varargin{2:end});
0197     %%%%%%%%%%% f = mfir(pzmodel, plist) %%%%%%%%%%%
0198     elseif isa(varargin{1}, 'pzmodel')
0199         pzm = varargin{1};
0200         if ~isa(pzm, 'pzmodel')
0201             error('### unknown constructor type for mfir.');
```

```

0202         end
0203         pl = varargin{2};
0204         if ~isa(pl, 'plist')
0205             error('### unknown constructor type for mfir.');
```

```

0206         end
0207
0208         pl = append(pl, param('pzmodel', pzm));
0209
0210         f = mfir(pl);
0211
0212     %%%%%%%%%%% f = mfir(ao, plist) %%%%%%%%%%%
0213     elseif isa(varargin{1}, 'ao')
0214
0215         if isa(varargin{2}, 'plist')
0216             pl = combine(varargin{2}, getDefaultPlist());
0217         else
0218             pl = getDefaultPlist();
0219         end
0220         f = mfirFromAO(ALGONAME, VERSION, varargin{1}, pl);
0221

```

```

0222     else
0223         error('### Unknown constructor method.');
```

```

0224     end
0225 otherwise
0226     error('### incorrect input arguments for mfir constructor.');
```

```

0227 end
0228
0229 end
0230
0231 %-----
0232 % create FIR filter from magnitude of input A0/fsdata
0233 function filt = mfirFromA0(ALGONAME, VERSION, a, pl)
0234
0235 %% Check that a.data is a fsdata object
0236 if ~isa(a.data, 'fsdata')
0237     error('### Please use an analysis object with a fsdata data object to
create a mfir object.');
```

```

0238 end
0239
0240 N      = find(pl, 'N');
0241 win    = find(pl, 'Win');
0242 method = find(pl, 'method');
```

```

0243
0244 fs = a.data.fs;
0245 f  = a.data.x;
0246 xx = abs(a.data.y);
0247 g  = 1; %median(xx);
0248
0249 ffm = f/(fs/2);
0250 switch method
0251     case 'frequency-sampling'
0252         % check window
0253         if length(win.win) ~= N+1
0254             warning('!!! resizing window function to match desired filter order.');
```

```

0255             if strcmp(win.name, 'Kaiser') || strcmp(win.name, 'Flattop')
0256                 win = specwin(win.name, N+1, win.psl1);
0257             else
0258                 win = specwin(win.name, N+1);
0259             end
0260         end
0261         disp('** designing filter using frequency-sampling method [help fir2]');
0262         mmaps = fir2(N, ffm, xx, win.win);
0263     case 'least-squares'
0264         error('### this design method is not working properly yet.');
```

```

0265         if mod(length(ffm),2)
0266             ffm = ffm(1:end-1);
0267             xx  = xx(1:end-1);
0268         end
0269         disp('** designing filter using least-squares method [help firls]');
0270         mmaps = firls(N, ffm, xx);
0271     case 'Parks-McClellan'
0272         error('### this design method is not working properly yet.');
```

```

0273         disp('** designing filter using Parks-McClellan method [help firpm]');
```

```

0274     mtaps = firpm(N, ffm, xx);
0275     otherwise
0276         error('### unknown filter design method.');
```

0277 end

```

0278
0279 % Make mfir object
0280 filt      = init(VERSION);
0281 filt.name  = sprintf('fir(%s)', a.name);
0282 filt.fs    = fs;
0283 filt.ntaps = length(mtaps);
0284 filt.a     = mtaps;
0285 filt.gd    = (filt.ntaps+1)/2;
0286 filt.gain  = g;
0287 filt.histout = zeros(1,filt.ntaps-1); % initialise output history
0288 filt.infile = '';
0289 filt.plist  = plist();
0290 filt.created = time;
0291 filt.version = VERSION;
0292 end
0293
0294 %-----
0295 % default plist
0296 function pl = getDefaultPlist()
0297
0298     N = 512;
0299     pl = plist(param('N', N));
0300     pl = append(pl, param('method', 'frequency-sampling'));
0301     pl = append(pl, param('Win', specwin('Hanning', N+1)));
0302
0303 end
0304
0305 %%%%%%%%%% define mfir properties %%%%%%%%%%
0306
0307 function f = init(VERSION)
0308     f.name      = 'None';
0309     f.fs        = 0;
0310     f.ntaps     = 0;
0311     f.a         = [];
0312     f.gd        = 0;
0313     f.gain      = 0;
0314     f.histout   = [];
0315     f.infile    = '';
0316     f.plist     = plist();
0317     f.created   = time;
0318     f.version   = VERSION;
0319     f          = class(f, 'mfir');
0320 end
0321
0322 %%%%%%%%%% Create mfir object %%%%%%%%%%
```

[Home](#) > [classes](#) > [@miir](#) > [private](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@provenance](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create a provenance object.

SYNOPSIS

```
function po = fromxml(xml)
```

DESCRIPTION

```
FROMXML read from part of an XML tree and create a provenance object.
```

```
M Hewitson 28-08-07
```

```
$Id: fromxml.m,v 1.1 2007/08/29 18:39:05 hewitson Exp $
```

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE provenance objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function po = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create a provenance object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.1 2007/08/29 18:39:05 hewitson Exp $
0008 %
0009
0010
```

```
0011 sch = xml.getChildNodes;
0012 for c=1:sch.getLength
0013     x = sch.item(c-1);
0014     switch char(x.getNodeName)
0015         case 'creator'
0016             p.creator = char(x.getTextContent);
0017         case 'created'
0018             xch = x.getChildNodes;
0019             p.created = time(xch.item(1));
0020         case 'ip'
0021             p.ip = char(x.getTextContent);
0022         case 'hostname'
0023             p.hostname = char(x.getTextContent);
0024         case 'os'
0025             p.os = char(x.getTextContent);
0026         case 'matlab_version'
0027             p.matlab_version = char(x.getTextContent);
0028         case 'sigproc_version'
0029             p.sigproc_version = char(x.getTextContent);
0030         case 'ltpda_version'
0031             p.ltpda_version = char(x.getTextContent);
0032         otherwise
0033     end
0034 end
0035
0036 % build provenance
0037 po = provenance(p);
0038
0039 % END
```

[Home](#) > [classes](#) > [@miir](#) > miir.m

miir

PURPOSE

MIIR IIR filter object class constructor.

SYNOPSIS

function f = miir(varargin)

DESCRIPTION

MIIR IIR filter object class constructor.

%%%

DESCRIPTION: MIIR IIR filter object class constructor.
Create an miir object.

CONSTRUCTOR: f = miir()
creates an empty miir object.

f = miir(fi)
creates a copy of the input miir object, fi.

f = miir('foo_iir.fil')
create an miir object from a LISO IIR .fil file.

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 0.3]>>
```

```
'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('gain', 2.0))
>> pl = append(pl, param('fs', 1000))
>> pl = append(pl, param('fc', 200))
>> f = miir(pl)
```

VERSION: \$Id: miir.m,v 1.22 2008/01/15 20:31:35 hewitson Exp \$

HISTORY: 09-02-2007 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [get](#) GET get miir properties.
- [miir](#) MIIR IIR filter object class constructor.
- [set](#) SET set a miir object property.
- [xmlparse](#) XMLPARSE creates a miir filter from a miir xml file.

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.

SUBFUNCTIONS

- [function f = init\(\)](#)

SOURCE CODE

```
0001 function f = miir(varargin)
0002 % MIIR IIR filter object class constructor.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MIIR IIR filter object class constructor.
0007 %             Create an miir object.
0008 %
0009 % CONSTRUCTOR: f = miir()
0010 %             creates an empty miir object.
0011 %
0012 %             f = miir(fi)
0013 %             creates a copy of the input miir object, fi.
0014 %
0015 %             f = miir('foo_iir.fil')
0016 %             create an miir object from a LISO IIR .fil file.
0017 %
0018 %             f = miir(pl)
0019 %             create an miir object from the description
0020 %             given in the parameter list.
0021 %
0022 % Parameters for standard filters:
0023 %
0024 %             'type'   - 'highpass', 'lowpass',
0025 %                       'bandpass', 'bandreject'.
0026 %                       <<default: 'lowpass'>>
0027 %             'gain'   - gain of filter
0028 %                       <<default: 1.0>>
0029 %             'fs'     - sample frequency to design for
0030 %                       <<default: 1Hz>>
0031 %             'order'  - order of filter
0032 %                       <<default: 1>>
0033 %             'fc'     - corner frequencies.
0034 %                       This is a two element vector for bandpass
0035 %                       and bandreject filters.
0036 %                       <<default: 0.25 or [0.1 0.3]>>
0037 %             'ripple' - Specify pass/stop-band ripple for bandpass
0038 %                       and bandreject filters
0039 %                       <<default: 0.5>>
0040 %
0041 % EXAMPLE 1: Create a default lowpass filter.
0042 %
0043 %             >> f = miir(plist())
0044 %
0045 % EXAMPLE 2: Create an order 1 highpass filter with high frequency gain 2.
0046 %             Filter is designed for 1kHz sampled data and has a cut-off
0047 %             frequency of 200Hz.
0048 %
0049 %             >> pl = plist();
0050 %             >> pl = append(pl, param('type', 'highpass'))
0051 %             >> pl = append(pl, param('gain', 2.0))
0052 %             >> pl = append(pl, param('fs', 1000))
0053 %             >> pl = append(pl, param('fc', 200))
0054 %             >> f = miir(pl)
0055 %
0056 % VERSION:      $Id: miir.m,v 1.22 2008/01/15 20:31:35 hewitson Exp $

```

```

0057 %
0058 % HISTORY:      09-02-2007 M Hewitson
0059 %              Creation
0060 %
0061 %%%%%%%%%%%%%%
0062
0063 VERSION = '$Id: miir.m,v 1.22 2008/01/15 20:31:35 hewitson Exp $';
0064
0065 %%%% Is this a 'Params' call? %%%%
0066 if nargin == 2 && isa(varargin{1}, 'miir') && ischar(varargin{2})
0067     if strcmp(varargin{2}, 'Params')
0068         f = plist;
0069         return
0070     elseif strcmp(varargin{2}, 'Version')
0071         f = VERSION;
0072         return
0073     end
0074 end
0075
0076 %%%%%%%%%%%%%% define miir properties %%%%%%%%%%%%%%
0077
0078 function f = init()
0079     f.name      = 'None';
0080     f.fs        = 0;
0081     f.ntaps     = 0;
0082     f.a         = [];
0083     f.b         = [];
0084     f.gain      = 0;
0085     f.histin    = [];
0086     f.histout   = [];
0087     f.infile    = '';
0088     f.plist     = plist();
0089     f.created   = time;
0090     f.version   = VERSION;
0091     f = class(f, 'miir');
0092 end
0093
0094 %%%%%%%%%%%%%% Create miir object %%%%%%%%%%%%%%
0095
0096 switch nargin
0097
0098     %%%%%%%%%%%%% h = miir() %%%%%%%%%%%%%%
0099     % create default miir object
0100     case 0
0101         f = init();
0102
0103     % Copy, LISO, filt, standard
0104     case 1
0105
0106         %%%%%%%%%%%%% Create from XML fragment %%%%%%%%%%%%%%
0107         if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0108             f = fromxml(varargin{1});
0109         %%%%%%%%%%%%% From File %%%%%%%%%%%%%%

```

```

0110     elseif ischar(varargin{1})
0111
0112         f = init();
0113         filename = varargin{1};
0114         [path, name, ext, vers] = fileparts(filename);
0115         switch ext
0116             case '.mat'
0117                 f = load(filename);
0118                 f = f.a;
0119             case '.xml'
0120                 f = xmlparse(miir, filename);
0121             case '.filt'
0122                 error('### .filt files are no longer supported. ');
0123                 % filt      = mfilload(filename);
0124                 % f.name    = filt.name;
0125                 % f.fs      = filt.fs;
0126                 % f.ntaps   = filt.ntaps;
0127                 % f.a       = filt.a;
0128                 % f.b       = filt.b;
0129                 % f.gain    = filt.gain;
0130                 % f.histin  = filt.histin;
0131                 % f.histout = filt.histout;
0132                 % f.infile  = filename;
0133             case '.fil'
0134                 filt      = filload(filename);
0135                 f.name    = filt.name;
0136                 f.fs      = filt.fs;
0137                 f.ntaps   = filt.ntaps;
0138                 f.a       = filt.a;
0139                 f.b       = filt.b;
0140                 f.gain    = filt.gain;
0141                 f.histin  = filt.histin;
0142                 f.histout = filt.histout;
0143                 f.infile  = filename;
0144             otherwise
0145                 error('### Unknown file type. ');
0146         end
0147
0148         %%%%%%%%% h = miir(miir) %%%%%%%%%
0149         % copy
0150     elseif isa(varargin{1}, 'miir')
0151         f = varargin{1};
0152         f.version = VERSION;
0153
0154         %%%%%%%%% h = miir(struct) %%%%%%%%%
0155         % Struct
0156     elseif isstruct(varargin{1})
0157
0158         f = init();
0159
0160         fields = fieldnames(varargin{1});
0161         for ii = 1:length(fields)

```

```

0162         field = fields{ii};
0163
0164         %%% plist -> plist-object
0165         if strcmp(field, 'plist')
0166             if isstruct(varargin{1}.(field))
0167                 f.(field) = plist(varargin{1}.(field));
0168             else
0169                 f.(field) = varargin{1}.(field);
0170             end
0171         %%% created -> time-object
0172         elseif strcmp(field, 'created')
0173             created = varargin{1}.created;
0174             if isstruct(created)
0175                 created = time(created);
0176             end
0177             f.created      = created;
0178         %%% All other
0179         else
0180             try
0181                 f.(field) = varargin{1}.(field);
0182             catch
0183                 error('### The field '%s' in the struct is not a mfir property.',
0184                     field)
0185             end
0186         end
0187
0188
0189         %%%%%%%%%%% h = miir(plist) %%%%%%%%%%%
0190         % Parameter list
0191         elseif isa(varargin{1}, 'plist')
0192
0193             pl    = varargin{1};
0194             type = find(pl, 'type');
0195             pzm   = find(pl, 'pzmodel');
0196
0197             f = init();
0198
0199             % deal with empty plist
0200             if isempty(type) && isempty(pzm)
0201                 type = 'lowpass';
0202             end
0203
0204             if ~isempty(type)
0205                 % check and fill parameter list
0206                 plo = parseFilterParams(pl);
0207                 switch type
0208                     case 'lowpass'
0209                         f = mklowpass(f, plo);
0210                     case 'highpass'
0211                         f = mkhighpass(f, plo);
0212                     case 'bandpass'
0213                         f = mkbandpass(f, plo);

```



```

0214         case 'bandreject'
0215             f = mkbandreject(f, plo);
0216         otherwise
0217             error('### unknown standard filter type in miir constructor.');
```

0218 end

```

0219         f.infile = '';
0220         f.plist  = plo;
0221
0222     elseif ~isempty(pzm)
0223
0224         f = init();
0225
0226         fs = find(pl, 'fs');
0227         if isempty(fs)
0228             % get max freq in pzmodel
0229             fs = 8*getupperFreq(pzm);
0230             warning([sprintf('!!! no sample rate specified. Designing for fs=%
2.2fHz.', fs)...
0231                     sprintf('\nThe filter will be redesigned later when
used.')]]);
0232
0233         end
0234         % make MIIR filter
0235         f = tomiir(pzm, fs);
0236         f = set(f, 'name', get(pzm, 'name'));
0237         f = set(f, 'plist', pl);
0238
0239     else
0240         error('### unknown constructor type for miir.');
```

0241 end

```

0242 else
0243     error('### unknown constructor type for miir.');
```

0244 end

```

0245
0246 %%%%%%%%%%% h = miir(pzmodel, plist) %%%%%%%%%%%
0247 case 2 % pzmodel followed by plist
0248
0249 %%%%%%%%%%% From DATABASE
0250 if isa(varargin{1}, 'database')
0251     f = retrieve(varargin{1}, varargin{2:end});
0252 else
0253     pzm = varargin{1};
0254     if ~isa(pzm, 'pzmodel')
0255         error('### unknown constructor type for miir.');
```

0256 end

```

0257     pl = varargin{2};
0258     if ~isa(pl, 'plist')
0259         error('### unknown constructor type for miir.');
```

0260 end

```

0261
0262     pl = append(pl, param('pzmodel', pzm));
0263
```

```

0264     f = miir(pl);
0265 end
0266
0267 %%%%%%%%%% h = miir(a, b, fs) %%%%%%%%%%
0268 case 3
0269
0270     % then we have 3 numeric inputs for a and b coefficients and fs
0271     a = varargin{1};
0272     b = varargin{2};
0273     fs = varargin{3};
0274     if ~isnumeric(a) || ~isnumeric(b) || ~isnumeric(fs)
0275         error('### unknown constructor type for miir.');

```

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@pzmodel](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create a pzmodel object.

SYNOPSIS

function po = fromxml(xml)

DESCRIPTION

FROMXML read from part of an XML tree and create a pzmodel object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.3 2007/10/15 17:16:57 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE pzmodel objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function po = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create a pzmodel object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.3 2007/10/15 17:16:57 ingo Exp $
0008 %
```

```

0009
0010
0011 % disp('----- pzmodel -----')
0012 % xml.getNodeName
0013 % disp('----- pzmodel -----')
0014
0015 % default
0016 p.name      = '';
0017 p.poles     = [];
0018 p.zeros     = [];
0019 p.gain      = 0;
0020 p.created   = '';
0021 p.version   = '';
0022
0023 sch = xml.getChildNodes;
0024 for c=1:sch.getLength
0025     x = sch.item(c-1);
0026     switch char(x.getNodeName)
0027         case 'name'
0028             p.name = char(x.getTextContent);
0029         case 'gain'
0030             p.gain = str2num(char(x.getTextContent));
0031         case 'poles'
0032             p.poles = [];
0033             xch = x.getChildNodes;
0034             for j=1:xch.getLength
0035                 it = xch.item(j-1);
0036                 if strcmp(char(it.getNodeName), 'pole')
0037                     p.poles = [p.poles pole(it)];
0038                 end
0039             end
0040         case 'zeros'
0041             p.zeros = [];
0042             xch = x.getChildNodes;
0043             for j=1:xch.getLength
0044                 it = xch.item(j-1);
0045                 if strcmp(char(it.getNodeName), 'zero')
0046                     p.zeros = [p.zeros zero(it)];
0047                 end
0048             end
0049         case 'created'
0050             xch = x.getChildNodes;
0051             p.created = time(xch.item(1));
0052         case 'version'
0053             p.version = char(x.getTextContent);

```

```
0054     otherwise
0055     end
0056 end
0057
0058 % build pzmodel
0059 po = pzmodel(p);
0060
0061 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@param](#) > [private](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > fromxml.m

fromxml

PURPOSE [^](#)

FROMXML read from part of an XML tree and create a specwin object.

SYNOPSIS [^](#)

function w = fromxml(xml)

DESCRIPTION [^](#)

FROMXML read from part of an XML tree and create a specwin object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.3 2008/01/11 14:05:13 ingo Exp \$

CROSS-REFERENCE INFORMATION [^](#)

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE specwin objects with given ids from the LTPDA repository specified by the input

SOURCE CODE [^](#)

```
0001 function w = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create a specwin object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.3 2008/01/11 14:05:13 ingo Exp $
0008 %
```

```
0009
0010
0011 sch = xml.getChildNodes;
0012 for c=1:sch.getLength
0013     x = sch.item(c-1);
0014     switch char(x.getNodeName)
0015         case 'created'
0016             created = char(x.getTextContent);
0017         case 'version'
0018             ver = char(x.getTextContent);
0019         case 'name'
0020             name = char(x.getTextContent);
0021         case 'psll'
0022             psll = str2num(char(x.getTextContent));
0023         case 'n'
0024             n = str2num(char(x.getTextContent));
0025         otherwise
0026     end
0027 end
0028
0029 % build window
0030 switch name
0031     case 'Kaiser'
0032         w = specwin(name, n, psll);
0033     otherwise
0034         w = specwin(name, n);
0035 end
0036
0037 w = set(w, 'created', created);
0038 w = set(w, 'version', ver);
0039
0040
0041 % END
```

[Home](#) > [classes](#) > [@time](#) > minus.m

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 1.4 2008/01/14 20:03:52 ingo Exp \$

HISTORY: 03-08-2007 Diepholz
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [set](#) SET set time properties.

This function is called by:

SOURCE CODE

```

0001 function res = minus(t1, t2)
0002 % MINUS overloads - operator for time objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MINUS overloads - operator for analysis objects.
0007 %             This function subtracts a time in seconds from a time object.
0008 %             It is possible to define the subtrahend as a string,
0009 %             number or a time object.
0010 %
0011 % CALL:       t2 = t1 - value (in seconds)
0012 %             sec = t1 - t2;
0013 %
0014 % EXAMPLES:   >> t1      = time('2007-07-01 12:23:33');
0015 %             >> t2      = t1 - 30;
0016 %             >> t3      = t1 - '30';
0017 %             >> seconds = t1 - t2;
0018 %
0019 % VERSION:    $Id: minus.m,v 1.4 2008/01/14 20:03:52 ingo Exp $
0020 %
0021 % HISTORY:    03-08-2007 Diepholz
0022 %             Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 VERSION = '$Id: minus.m,v 1.4 2008/01/14 20:03:52 ingo Exp $';
0027
0028 % 'Params' Call
0029 if nargin == 2
0030     if isa(t1, 'time') && strcmp(t2, 'Params')
0031         res = plist();
0032         return
0033     elseif isa(t1, 'time') && strcmp(t2, 'Version')
0034         res = VERSION;
0035         return
0036     end
0037 end
0038
0039 if isnumeric(t2)
0040     t2 = ltpda_mat2str(t2);
0041 end
0042
0043 %%%>> t2 = t1 - 30; %%%
0044 %%%>> t2 = t1 - '30'; %%%
0045 if ischar(t2)
0046     java_format_str_seconds = 'ss';
0047
0048     utc_timezone = java.util.TimeZone.getTimeZone('UTC');
0049

```

```
0050     t_format_seconds = java.text.SimpleDateFormat(java_format_str_seconds);
0051     t_format_seconds.setTimeZone(utc_timezone);
0052
0053     epoch_time_seconds = t_format_seconds.parse(t2);
0054     epoch_time_seconds = epoch_time_seconds.getTime;
0055
0056     res = set(t1, 'utc_epoch_milli', t1.utc_epoch_milli-epoch_time_seconds);
0057
0058     %%%%    >> number = t1 - t2;    %%%%
0059 elseif isa(t2, 'time')
0060
0061     res = (t1.utc_epoch_milli - t2.utc_epoch_milli)/1000;
0062
0063 else
0064     error('### Unknown minus-operation to a time object.');
```

[Home](#) > [classes](#) > [@history](#) > string.m

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 1.3 2007/12/05 16:26:22 ingo 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

```

0001 function cmd = string(hs, varargin)
0002 % STRING writes a command string that can be used to recreate the input history
0003 % object.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0007 %               input history object.
0008 %
0009 % CALL:         cmd = string(history_obj)
0010 %
0011 % INPUT:        history_obj - history object
0012 %
0013 % OUTPUT:       cmd          - command string to create the input object
0014 %
0015 % VERSION:      $Id: string.m,v 1.3 2007/12/05 16:26:22 ingo Exp $
0016 %
0017 % HISTORY:      29-03-2007 M Hewitson
0018 %               Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: string.m,v 1.3 2007/12/05 16:26:22 ingo Exp $';
0023
0024 % Check if this is a call for parameters
0025 if nargin == 2
0026     if isa(hs, 'history') && ischar(varargin{1})
0027         in = char(varargin{1});
0028         if strcmp(in, 'Params')
0029             cmd = plist;
0030             return
0031         elseif strcmp(in, 'Version')
0032             cmd = VERSION;
0033             return
0034         end
0035     end
0036 end
0037
0038 cmd = '[';
0039
0040 for j=1:length(hs)
0041     h = hs(j);
0042     name = get(h, 'name');
0043     vers = get(h, 'version');
0044     pl    = get(h, 'plist');
0045     hi    = get(h, 'inhists');
0046
0047     if isempty(pl)
0048         plstr = '[]';

```

```
0049     else
0050         plstr = string(pl);
0051     end
0052     if isempty(plstr)
0053         plstr = '[]';
0054     end
0055
0056     if isempty(hi)
0057         histr = '[]';
0058     else
0059         histr = string(hi);
0060     end
0061     if isempty(histr)
0062         histr = '[]';
0063     end
0064
0065     cmd = [cmd 'history(' ...
0066           '    name ' ...
0067           '    vers ' ...
0068           plstr ', ' ...
0069           histr ...
0070           ') '];
0071 end
0072
0073 cmd = [cmd '']];
0074
0075 if strcmp(cmd, '[]')
0076     cmd = '';
0077 end
0078
0079 % END
```


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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@time](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create a time object.

SYNOPSIS

function t = fromxml(xml)

DESCRIPTION

FROMXML read from part of an XML tree and create a time object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.2 2007/10/16 17:37:21 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE time objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function t = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create a time object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.2 2007/10/16 17:37:21 ingo Exp $
0008 %
0009
0010 % disp('----- time -----')
0011 % xml.getNodeName
0012 % disp('----- time -----')
0013
0014 sch = xml.getChildNodes;
0015 for c=1:sch.getLength
0016     x = sch.item(c-1);
```

```

0017  switch char(x.getNodeName)
0018      case 'name'
0019          name = char(x.getTextContent);
0020      case 'utc_epoch_milli'
0021          utc_epoch_milli = str2num(char(x.getTextContent));
0022      case 'timezone'
0023          timezone = char(x.getTextContent);
0024      case 'timeformat'
0025          xch = x.getChildNodes;
0026          tf = timeformat(xch.item(1));
0027      case 'time_str'
0028          time_str = char(x.getTextContent);
0029      case 'version'
0030          version = char(x.getTextContent);
0031      case 'created'
0032          created = char(x.getTextContent);
0033      otherwise
0034  end
0035 end
0036
0037 % build time
0038 t = time(utc_epoch_milli);
0039 t = set(t, 'timezone',    timezone);
0040 t = set(t, 'timeformat', tf);
0041 % t = set(t, 'time_str',    time_str); % obsolete because utc_epoch_milli is set
0042 t = set(t, 'name',      name);
0043 t = set(t, 'version',    version);
0044 t = set(t, 'created',    created);
0045 % t = set(t, 'timeformat', tf);
0046
0047 % END

```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > mk_test_objs.m

mk_test_objs

PURPOSE

SYNOPSIS

This is a script file.

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 a1 = ao(plist('waveform', 'sine wave', 'f', 1, 'phi', 0, 'nsecs', 10, 'fs',  
100));  
0002 a2 = ao  
0003  
0004 pl = plist('a', 1, 'b', 2)  
0005 p1 = param('c', 3);  
0006 p2 = param
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@mfir](#) > string.m

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 1.3 2007/12/10 18:07:17 ingo 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

```
0001 function cmd = string(f, varargin)
0002 % STRING writes a command string that can be used to recreate the input filter
```

```

object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0007 %                input filter object.
0008 %
0009 % CALL:          cmd = string(mfir)
0010 %
0011 % VERSION:       $Id: string.m,v 1.3 2007/12/10 18:07:17 ingo Exp $
0012 %
0013 % HISTORY:       29-03-2007 M Hewitson
0014 %                Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: string.m,v 1.3 2007/12/10 18:07:17 ingo Exp $';
0019
0020 % Check if this is a call for parameters
0021 if nargin == 2
0022     if isa(f, 'mfir') && ischar(varargin{1})
0023         in = char(varargin{1});
0024         if strcmp(in, 'Params')
0025             cmd = plist;
0026             return
0027         elseif strcmp(in, 'Version')
0028             cmd = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 infile = get(f, 'infile');
0035 if ~isempty(infile)
0036     cmd = ['mfir('' infile '')'];
0037 else
0038     pl = get(f, 'plist');
0039     cmd = ['mfir(' string(pl) ')'];
0040 end
0041
0042
0043
0044 % END

```

[Home](#) > [classes](#) > [@plist](#) > [private](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@timeformat](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create a timeformat object.

SYNOPSIS

```
function tf = fromxml(xml)
```

DESCRIPTION

```
FROMXML read from part of an XML tree and create a timeformat object.
```

```
M Hewitson 28-08-07
```

```
$Id: fromxml.m,v 1.1 2007/08/29 18:40:52 hewitson Exp $
```

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE timeformat objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function tf = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create a timeformat object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.1 2007/08/29 18:40:52 hewitson Exp $
0008 %
0009
0010 %
```

```
0011 % disp('----- timeformat -----')
0012 % xml.getNodeName
0013 % disp('----- timeformat -----')
0014
0015 pl.params = [];
0016 sch = xml.getChildNodes;
0017 for c=1:sch.getLength
0018     x = sch.item(c-1);
0019     switch char(x.getNodeName)
0020         case 'format_str'
0021             format_str = char(x.getTextContent);
0022         case 'format_nr'
0023             format_nr = str2num(char(x.getTextContent));
0024         case 'default_matlab_str'
0025             default_matlab_str = char(x.getTextContent);
0026         case 'default_matlab_nr'
0027             default_matlab_nr = str2num(char(x.getTextContent));
0028         case 'default_java_str'
0029             default_java_str = char(x.getTextContent);
0030         otherwise
0031     end
0032 end
0033
0034 % build timeformat
0035 tf = timeformat(format_str,...
0036                 default_matlab_str,...
0037                 default_matlab_nr,...
0038                 default_java_str);
0039
0040 % END
```

[Home](#) > [classes](#) > [@mfir](#) > [private](#) > mkbandpass.m

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

```
0001 function f = mkbandpass(f, pl)
0002 % MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MKBANDPASS return a bandpass filter miir().
0007 %           A Cheby filter is used.
0008 %
```

```

0009 % CALL:          f = mkbandpass(f, pl)
0010 %
0011 % VERSION: $Id: mkbandpass.m,v 1.4 2007/10/24 10:55:11 ingo Exp $
0012 %
0013 % HISTORY: 27-08-2002 M Hewitson
0014 %          Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 g      = find(pl, 'gain');
0019 fc      = find(pl, 'fc');
0020 fs      = find(pl, 'fs');
0021 order   = find(pl, 'order');
0022 win     = find(pl, 'win');
0023
0024 if(fc > fs/2)
0025     error('fc must be < fs/2');
0026 end
0027 if(fc(1) > fc(2))
0028     error('fc(1) must be < fc(2)');
0029 end
0030
0031 f.name   = 'std bandpass';
0032 f.fs     = fs;
0033 f.ntaps  = order+1;
0034 f.a      = fir1(order, 2.*fc/fs, 'bandpass', win.win);
0035 f.gd     = (f.ntaps)/2;
0036 f.gain   = g;
0037 f.histout = zeros(1,f.ntaps-1); % initialise output history
0038
0039 % END
0040

```

[Home](#) > [classes](#) > [@miir](#) > string.m

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 1.3 2007/12/10 18:07:17 ingo 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

```

0001 function cmd = string(f, varargin)
0002 % STRING writes a command string that can be used to recreate the input filter
object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0007 %               input filter object.
0008 %
0009 % CALL:         cmd = string(miir)
0010 %
0011 % INPUTS:       cmd - IIR filter object
0012 %
0013 % OUTPUTS:      miir - command string to create the IIR filter object
0014 %
0015 % VERSION:      $Id: string.m,v 1.3 2007/12/10 18:07:17 ingo Exp $
0016 %
0017 % HISTORY:      29-03-2007 M Hewitson
0018 %               Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: string.m,v 1.3 2007/12/10 18:07:17 ingo Exp $';
0023
0024 % Check if this is a call for parameters
0025 if nargin == 2
0026     if isa(f, 'miir') && ischar(varargin{1})
0027         in = char(varargin{1});
0028         if strcmp(in, 'Params')
0029             cmd = plist;
0030             return
0031         elseif strcmp(in, 'Version')
0032             cmd = VERSION;
0033             return
0034         end
0035     end
0036 end
0037
0038 infile = get(f, 'infile');
0039 if ~isempty(infile)
0040     cmd = ['miir('' infile ''')'];
0041 else
0042     pl = get(f, 'plist');
0043     cmd = ['miir(' string(pl) ')'];
0044 end
0045
0046
0047
0048 % END

```


Contents

PURPOSE

%% class: pole %%%

SYNOPSIS

This is a script file.

DESCRIPTION

%% class: pole %%%

[pole/cat](matlab:help pole/cat) - concatonate poles into a vector.

[pole/char](matlab:help pole/char) - convert a pole object into a string.

[pole/cp2iir](matlab:help pole/cp2iir) - Return a,b IIR filter coefficients for a complex pole designed using the bilinear transform.

[pole/display](matlab:help pole/display) - overloads display functionality for pole objects.

[pole/eq](matlab:help pole/eq) - overloads the == operator for pole objects.

[pole/get](matlab:help pole/get) - get pole properties.

[pole/isfield](matlab:help pole/isfield) - tests if the given field is one of the object properties.

[pole/ne](matlab:help pole/ne) - overloads the ~= operator for pole objects.

[pole/pole](matlab:help pole/pole) - construct a pole object.

[pole/resp](matlab:help pole/resp) - returns the complex response of the pole object.

[pole/rp2iir](matlab:help pole/rp2iir) - Return a,b coefficients for a real pole designed using the bilinear transform.

[pole/save](matlab:help pole/save) - a pole object to file.

[pole/set](matlab:help pole/set) - set a pole property.

[pole/string](matlab:help pole/string) - writes a command string that can be used to recreate the input pole object.

[pole/subsref](matlab:help pole/subsref) - Define field name indexing for pole objects.

[pole/xml](matlab:help pole/xml) - convert a pole object to an DOM XML object.

[pole/xmladd](matlab:help pole/xmladd) - add a pole object to the given node of the input XML DOM

[pole/xmlparse](matlab:help pole/xmlparse) - creates a pole from a pole xml file.

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@timespan](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create a timespan object.

SYNOPSIS

```
function t = fromxml(xml)
```

DESCRIPTION

FROMXML read from part of an XML tree and create a timespan object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.2 2007/10/16 17:37:21 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE timespan objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function t = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create a timespan object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.2 2007/10/16 17:37:21 ingo Exp $
0008 %
0009
```

```

0010 % disp('----- timespan -----')
0011 % xml.getNodeName
0012 % disp('----- timespan -----')
0013
0014 sch = xml.getChildNodes;
0015 for c=1:sch.getLength
0016     x = sch.item(c-1);
0017     switch char(x.getNodeName)
0018         case 'name'
0019             name = char(x.getTextContent);
0020         case 'start'
0021             xch = x.getChildNodes;
0022             start = time(xch.item(1));
0023         case 'end'
0024             xch = x.getChildNodes;
0025             stop = time(xch.item(1));
0026         case 'timeformat'
0027             xch = x.getChildNodes;
0028             tf = timeformat(xch.item(1));
0029         case 'timezone'
0030             timezone = char(x.getTextContent);
0031         case 'version'
0032             version = char(x.getTextContent);
0033         case 'created'
0034             xch = x.getChildNodes;
0035             created = time(xch.item(1));
0036         otherwise
0037     end
0038 end
0039
0040 % build time
0041 t = timespan(start, stop, tf);
0042 t = set(t, 'name', name);
0043 t = set(t, 'timezone', timezone);
0044 t = set(t, 'version', version);
0045 t = set(t, 'created', created);
0046
0047
0048 % END

```

[Home](#) > [classes](#) > [@miir](#) > [private](#) > mkbandpass.m

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

```
0001 function f = mkbandpass(f, pl)
0002 % MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MKBANDPASS return a bandpass filter miir(). A Cheby filter is
0007 % used.
0008 % CALL:            f = mkbandpass(f, pl)
```

```

0009 %
0010 % VERSION: $Id: mkbandpass.m,v 1.5 2007/10/24 10:55:11 ingo Exp $
0011 %
0012 % HISTORY: 27-08-2002 M Hewitson
0013 %           Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 g      = find(pl, 'gain');
0018 fc     = find(pl, 'fc');
0019 fs     = find(pl, 'fs');
0020 order  = find(pl, 'order');
0021 ripple = find(pl, 'ripple');
0022
0023 if(fc > fs/2)
0024     error('fc must be < fs/2');
0025 end
0026 if(fc(1) > fc(2))
0027     error('fc(1) must be < fc(2)');
0028 end
0029
0030
0031 f.name    = 'std bandpass';
0032 f.fs     = fs;
0033 f.ntaps   = 2*order+1;
0034 [f.a, f.b] = cheby1(order, ripple, 2.*fc./fs);
0035 f.gain    = g;
0036 f.histin  = zeros(1,f.ntaps-1);    % initialise input history
0037 f.histout = zeros(1,f.ntaps-1);    % initialise output history
0038
0039 % END
0040

```

[Home](#) > [classes](#) > [@pole](#) > [private](#) > Contents.m

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
complex 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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@tsdata](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create a tsdata object.

SYNOPSIS

```
function data = fromxml(xml)
```

DESCRIPTION

FROMXML read from part of an XML tree and create a tsdata object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.1 2007/08/29 18:41:53 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE tsdata objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function data = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create a tsdata object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.1 2007/08/29 18:41:53 hewitson Exp $
```

```

0008 %
0009
0010 % defaults
0011 xunits = '';
0012 yunits = '';
0013 dtype = '';
0014
0015 sch = xml.getChildNodes;
0016 for c=1:sch.getLength
0017     x = sch.item(c-1);
0018     switch char(x.getNodeName)
0019         case 'name'
0020             name = char(x.getTextContent);
0021         case 'xunits'
0022             xunits = char(x.getTextContent);
0023         case 'yunits'
0024             yunits = char(x.getTextContent);
0025         case 'version'
0026             version = char(x.getTextContent);
0027         case 'fs'
0028             fs = str2num(char(x.getTextContent));
0029         case 'nsecs'
0030             nsecs = str2num(char(x.getTextContent));
0031         case 't0'
0032             xch = x.getChildNodes;
0033             t0 = time(xch.item(1));
0034         case 'created'
0035             xch = x.getChildNodes;
0036             created = time(xch.item(1));
0037         case 'shape'
0038             shape = str2num(char(x.getTextContent));
0039         case 'dtype'
0040             dtype = char(x.getTextContent);
0041         case 'data'
0042             idata = ltpda_xml_read_datasamples(x, dtype);
0043         otherwise
0044     end
0045 end
0046
0047 % build fsdata
0048 data = tsdata();
0049 data = set_xy_axis (data, idata.x, idata.y);
0050 data = set(data, 'fs', fs);
0051 data = set(data, 't0', t0);
0052 data = set(data, 'nsecs', nsecs);

```



```
0053
0054 data = set(data, 'name', name);
0055 data = set(data, 'version', version);
0056 data = set(data, 'created', created);
0057 data = set(data, 'xunits', xunits);
0058 data = set(data, 'yunits', yunits);
0059
0060
0061 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

mkbandreject

PURPOSE

MKBANDREJECT return a low pass filter miir(). A Butterworth filter is used.

SYNOPSIS

function f = mkbandreject(f, pl)

DESCRIPTION

MKBANDREJECT return a low pass filter miir(). A Butterworth filter is used.

%%%

DESCRIPTION: MKBANDREJECT return a low pass filter miir().
A Butterworth filter is used.

CALL: f = mkbandreject(f, pl)

VERSION: \$Id: mkbandreject.m,v 1.4 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

```
0001 function f = mkbandreject(f, pl)
0002 % MKBANDREJECT return a low pass filter miir(). A Butterworth filter is used.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MKBANDREJECT return a low pass filter miir().
0007 %           A Butterworth filter is used.
0008 %
```

```

0009 % CALL:          f = mkbandreject(f, pl)
0010 %
0011 % VERSION: $Id: mkbandreject.m,v 1.4 2007/10/24 10:55:11 ingo Exp $
0012 %
0013 % HISTORY: 27-08-2002 M Hewitson
0014 %          Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 g      = find(pl, 'gain');
0019 fc     = find(pl, 'fc');
0020 fs     = find(pl, 'fs');
0021 order  = find(pl, 'order');
0022 win    = find(pl, 'Win');
0023
0024 if(fc > fs/2)
0025     error('fc must be < fs/2');
0026 end
0027 if(fc(1) > fc(2))
0028     error('fc(1) must be < fc(2)');
0029 end
0030
0031 f.name   = 'std bandreject';
0032 f.fs     = fs;
0033 f.ntaps  = order+1;
0034 f.a      = fir1(order, 2.*fc/fs, 'stop', win.win);
0035 f.gd     = (f.ntaps)/2;
0036 f.gain   = g;
0037 f.histout = zeros(1,f.ntaps-1); % initialise output history
0038
0039 % END
0040

```

[Home](#) > [classes](#) > [@pole](#) > string.m

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 1.5 2008/01/03 15:33:48 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

```
0001 function cmd = string(ps, varargin)
0002 % STRING writes a command string that can be used to recreate the input pole
object.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0007 %               input pole object.
0008 %
0009 % CALL:         cmd = string(pole);
0010 %
0011 % VERSION:      $Id: string.m,v 1.5 2008/01/03 15:33:48 ingo Exp $
0012 %
0013 % HISTORY:      02-04-2007 M Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: string.m,v 1.5 2008/01/03 15:33:48 ingo Exp $';
0019
0020 %%%% 'Params' && 'Version' Call %%%%
0021 if nargin == 2
0022     if isa(ps, 'pole') && ischar(varargin{1})
0023         in = varargin{1};
0024         if strcmp(in, 'Params')
0025             cmd = plist();
0026             return
0027         elseif strcmp(in, 'Version')
0028             cmd = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 cmd = '[';
0035 for j=1:length(ps)
0036     p = ps(j);
0037
0038     pl = plist();
0039     fields = fieldnames(p);
0040     for ii = 1:length(fields)
0041         field = fields{ii};
0042         if strcmp(field, 'ri')
0043             % do not add this field to the parametr list
0044         else
0045             pl = append(pl, field, p.(field));
0046         end
0047     end
0048
0049     cmd = [cmd ' pole(' string(pl) ')'];
0050 end
0051 cmd = [cmd ']';
0052
0053 % END

```


[Home](#) > [classes](#) > [@provenance](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@xydata](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create a xydata object.

SYNOPSIS

function data = fromxml(xml)

DESCRIPTION

FROMXML read from part of an XML tree and create a xydata object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.2 2007/09/03 09:34:50 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE xydata objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function data = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create a xydata object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.2 2007/09/03 09:34:50 hewitson Exp $
```

```

0008 %
0009
0010 % defaults
0011 xunits  = '';
0012 yunits  = '';
0013 dtype   = '';
0014 idata.x = [];
0015 idata.y = [];
0016
0017 sch = xml.getChildNodes;
0018 for c=1:sch.getLength
0019     x = sch.item(c-1);
0020     switch char(x.getNodeName)
0021         case 'name'
0022             name = char(x.getTextContent);
0023         case 'xunits'
0024             xunits = char(x.getTextContent);
0025         case 'yunits'
0026             yunits = char(x.getTextContent);
0027         case 'version'
0028             version = char(x.getTextContent);
0029         case 'created'
0030             xch = x.getChildNodes;
0031             created = time(xch.item(1));
0032         case 'shape'
0033             shape = str2num(char(x.getTextContent));
0034         case 'dtype'
0035             dtype = char(x.getTextContent);
0036         case 'data'
0037             idata = ltpda_xml_read_datasamples(x, dtype);
0038         otherwise
0039     end
0040 end
0041
0042 % build xydata
0043 data = xydata(idata.x, idata.y);
0044
0045 data = set(data, 'name', name);
0046 data = set(data, 'version', version);
0047 data = set(data, 'created', created);
0048 data = set(data, 'xunits', xunits);
0049 data = set(data, 'yunits', yunits);
0050
0051
0052 % END

```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@miir](#) > [private](#) > mkbandreject.m

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

```
0001 function f = mkbandreject(f, pl)
0002 % MKBANDREJECT return a low pass filter miir(). A Butterworth filter is used.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MKBANDREJECT return a low pass filter miir().
0007 %                    A Butterworth filter is used.
0008 %
```

```

0009 % CALL:          f = mkbandreject(f, pl)
0010 %
0011 % VERSION: $Id: mkbandreject.m,v 1.7 2008/01/02 17:54:22 ingo Exp $
0012 %
0013 % HISTORY: 27-08-2002 M Hewitson
0014 %          Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 g      = find(pl, 'gain');
0019 fc     = find(pl, 'fc');
0020 fs     = find(pl, 'fs');
0021 order  = find(pl, 'order');
0022 ripple = find(pl, 'ripple');
0023
0024 if(fc > fs/2)
0025     error('fc must be < fs/2');
0026 end
0027 if(fc(1) > fc(2))
0028     error('fc(1) must be < fc(2)');
0029 end
0030
0031 f.name    = 'std bandreject';
0032 f.fs      = fs;
0033 f.ntaps   = 2*order+1;
0034 [f.a, f.b] = cheby1(order, ripple, 2.*fc./fs, 'stop');
0035 f.gain    = g;
0036 f.histin  = zeros(1,f.ntaps-1); % initialise input history
0037 f.histout = zeros(1,f.ntaps-1); % initialise output history
0038
0039 % END
0040

```

[Home](#) > [classes](#) > [@pzmodel](#) > string.m

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 1.5 2008/01/08 14:57:07 ingo 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

```
0001 function cmd = string(p, varargin)
0002 % STRING writes a command string that can be used to recreate the input pzmodel
```

```

object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: STRING writes a command string that can be used to recreate
0007 %               the input pzmodel object.
0008 %
0009 % CALL:         cmd = string(pzmodel)
0010 %
0011 % VERSION:      $Id: string.m,v 1.5 2008/01/08 14:57:07 ingo Exp $
0012 %
0013 % HISTORY:      02-04-2007 M Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: string.m,v 1.5 2008/01/08 14:57:07 ingo Exp $';
0019
0020 % Check if this is a call for parameters or for the cvs-version number
0021 if nargin == 2
0022     if isa(p, 'pzmodel') && ischar(varargin{1})
0023         in = char(varargin{1});
0024         if strcmp(in, 'Params')
0025             cmd = plist();
0026             return
0027         elseif strcmp(in, 'Version')
0028             cmd = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 pl = plist();
0035 fields = fieldnames(p);
0036 for ii = 1:length(fields)
0037     field = fields{ii};
0038     pl = append(pl, field, p.(field));
0039 end
0040
0041 cmd = ['pzmodel(' string(pl) ')'];
0042
0043 % END

```

[Home](#) > [classes](#) > [@provenance](#) > [private](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@zero](#) > [private](#) > fromxml.m

fromxml

PURPOSE

FROMXML read from part of an XML tree and create a zero object.

SYNOPSIS

```
function po = fromxml(xml)
```

DESCRIPTION

FROMXML read from part of an XML tree and create a zero object.

M Hewitson 28-08-07

\$Id: fromxml.m,v 1.2 2007/10/16 17:37:21 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [retrieve](#) RETRIEVE zero objects with given ids from the LTPDA repository specified by the input

SOURCE CODE

```
0001 function po = fromxml(xml)
0002
0003 % FROMXML read from part of an XML tree and create a zero object.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007 % $Id: fromxml.m,v 1.2 2007/10/16 17:37:21 ingo Exp $
```

```

0008 %
0009
0010
0011 % disp('----- pole -----')
0012 % xml.getNodeName
0013 % disp('----- pole -----')
0014
0015 sch = xml.getChildNodes;
0016 for c=1:sch.getLength
0017     x = sch.item(c-1);
0018     switch char(x.getNodeName)
0019         case 'version'
0020             p.version = char(x.getTextContent);
0021         case 'name'
0022             p.name = char(x.getTextContent);
0023         case 'f'
0024             p.f = str2num(char(x.getTextContent));
0025         case 'q'
0026             p.q = str2num(char(x.getTextContent));
0027         case 'created'
0028             xch = x.getChildNodes;
0029             p.created = time(xch.item(1));
0030         otherwise
0031     end
0032 end
0033
0034 % build zero
0035 po = zero(p.f, p.q);
0036 po = set(po, 'name', p.name);
0037 po = set(po, 'version', p.version);
0038 po = set(po, 'created', p.created);
0039
0040 % END

```

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

```
0001 function f = mkhighpass(f, pl)
0002 % MKHIGHPASS return a high pass filter miir(). A Butterworth filter is used.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MKHIGHPASS return a high pass filter miir().
0007 %           A Butterworth filter is used.
0008 %
```

```

0009 % CALL:          f = mkhighpass(f, pl)
0010 %
0011 % VERSION: $Id: mkhighpass.m,v 1.4 2007/10/24 10:55:11 ingo Exp $
0012 %
0013 % HISTORY: 27-08-2002 M Hewitson
0014 %          Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 g      = find(pl, 'gain');
0019 fc      = find(pl, 'fc');
0020 fs      = find(pl, 'fs');
0021 order = find(pl, 'order');
0022 win     = find(pl, 'Win');
0023
0024 if(fc(1) > fs/2)
0025     error('fc must be < fs/2');
0026 end
0027
0028
0029 f.name      = 'std highpass';
0030 f.fs        = fs;
0031 f.ntaps     = order+1;
0032 f.a         = fir1(order, 2*fc/fs, 'high', win.win);
0033 f.gd        = (f.ntaps)/2;
0034 f.gain      = g;
0035 f.histout   = zeros(1,f.ntaps-1); % initialise output history
0036
0037
0038 % END
0039

```

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.

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

DESCRIPTION: STRING writes a command string that can be used to recreate the
             input window object.

CALL:       cmd = string(sw)

VERSION:    $Id: string.m,v 1.3 2008/01/11 14:04:43 ingo Exp $

HISTORY:    29-03-2007 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

```
0001 function cmd = string(sw ,varargin)
0002 % STRING writes a command string that can be used to recreate the input window
0003 % object.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: STRING writes a command string that can be used to recreate the
```

```

0007 %           input window object.
0008 %
0009 % CALL:      cmd = string(sw)
0010 %
0011 % VERSION:    $Id: string.m,v 1.3 2008/01/11 14:04:43 ingo Exp $
0012 %
0013 % HISTORY:    29-03-2007 M Hewitson
0014 %             Creation
0015 %
0016 %%%%%%%%%%%
0017
0018 VERSION = '$Id: string.m,v 1.3 2008/01/11 14:04:43 ingo Exp $';
0019
0020 % Check if this is a call for parameters
0021 if nargin == 2
0022     if isa(sw, 'specwin') && ischar(varargin{1})
0023         in = char(varargin{1});
0024         if strcmp(in, 'Params')
0025             cmd = plist();
0026             return
0027         elseif strcmp(in, 'Version')
0028             cmd = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 switch sw.name
0035     case 'Kaiser'
0036         cmd = ['specwin('' sw.name '', ' num2str(length(sw.win)) ', ' num2str(sw.
psll) ')'];
0037     case 'Flattop' % Flattop windows are not supported
0038         cmd = ['specwin('' sw.name '', ' num2str(length(sw.win)) ', ' num2str(sw.
psll) ')'];
0039     otherwise
0040         cmd = ['specwin('' sw.name '', ' num2str(length(sw.win)) ')'];
0041 end
0042
0043 % END

```

Contents

PURPOSE

%% class: pzmodel %%%

SYNOPSIS

This is a script file.

DESCRIPTION

%% class: pzmodel %%%

pzmodel/char	- convert a pole object into a string.
pzmodel/display	- overloads display functionality for pzmodel objects.
pzmodel/eq	- overloads the == operator for pzmodel objects.
pzmodel/get	- get a pole/zero model property.
pzmodel/getlowerFreq	- gets the frequency of the lowest pole or zero in the model.
pzmodel/getupperFreq	- gets the frequency of the highest pole or zero in the model.
pzmodel/isfield	- tests if the given field is one of the object properties.
pzmodel/ne	- overloads the ~= operator for pzmodel objects.
pzmodel/pzmodel	- constructor for pzmodel class.
pzmodel/resp	- returns the complex response of a pzmodel as an Analysis Object.
pzmodel/save	- a pzmodel object to file.
pzmodel/set	- sets a pole/zero model property.
pzmodel/string	- writes a command string that can be used to recreate the input pzmodel object.
pzmodel/subsref	- Define field name indexing for pole/zero model objects.
pzmodel/tomfir	- approximates a pole/zero model with an FIR filter.
pzmodel/tomiir	- converts a pzmodel to an IIR filter using a bilinear transform.
pzmodel/xml	- convert a pzmodel

object to an DOM XML object.

pzmodel/xmladd	- add a pzmodel
object to the given node of the input XML DOM	
pzmodel/xmlparse	- creates a
pzmodel from a pzmodel xml file.	

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

fsdata

PURPOSE

FSDATA frequency-series object class constructor.

SYNOPSIS

function fsd = fsdata(varargin)

DESCRIPTION

FSDATA frequency-series object class constructor.

%%%

DESCRIPTION: FSDATA frequency-series object class constructor.
Create a frequency-series data object.

Properties:

name	- name of frequency-series object
x	- frequency samples vector
y	- y samples vector
enbw	- equivalent noise bandwidth
navs	- number of averages
fs	- sample rate of data
xunits	- units to interpret the frequency samples (e.g., seconds)
yunits	- units to interpret the data samples (e.g., Volts)
version	- version of the constructor code
created	- creation time of this fsdata object.

Possible constructors:

fsd = fsdata()	- creates a blank frequency-series object
fsd = fsdata(y)	- creates a frequency-series object with the given y-data. Sample rate of the data is assumed to be 1Hz.
fsd = fsdata(f,y)	- creates a frequency-series object with the given (x,y)-data. The sample rate is then set as 2*x(end).
fsd = fsdata(y,fs)	- creates a frequency-series object with the given y-data and sample rate. The frequency vector is grown assuming the first y sample corresponds to 0Hz and the last sample corresponds to the Nyquist frequency.
fsd = fsdata(x,y,fs)	- creates a frequency-series object with the given x,y-data and sample rate.

HISTORY: 30-01-2007 Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [fsdata](#) FSDATA frequency-series object class constructor.
- [reshapeF](#) RESHAPEF reshape the frequency vector to match the y vector.
- [setFreq](#) SETFREQ sets the frequency vector of a fsdata object based on the length of the y vector and the sample rate.
- [xmlparse](#) XMLPARSE creates a fsdata from a fsdata xml file.

This function is called by:

- [fsdata](#) FSDATA frequency-series object class constructor.

SUBFUNCTIONS

- [function fs = init\(\)](#)

SOURCE CODE

```

0001 function fsd = fsdata(varargin)
0002 % FSDATA frequency-series object class constructor.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: FSDATA frequency-series object class constructor.
0007 %               Create a frequency-series data object.
0008 %
0009 %       Properties:
0010 %       name      - name of frequency-series object
0011 %       x         - frequency samples vector
0012 %       y         - y samples vector
0013 %       enbw      - equivalent noise bandwidth
0014 %       navs      - number of averages
0015 %       fs        - sample rate of data
0016 %       xunits    - units to interpret the frequency samples (e.g., seconds)
0017 %       yunits    - units to interpret the data samples      (e.g., Volts)
0018 %       version   - version of the constructor code
0019 %       created   - creation time of this fsdata object.
0020 %
0021 %       Possible constructors:
0022 %       fsd = fsdata()           - creates a blank frequency-series object
0023 %       fsd = fsdata(y)         - creates a frequency-series object with the given

```

```

0024 % y-data. Sample rate of the data is assumed to
0025 % be 1Hz.
0026 % fsd = fsdata(f,y) - creates a frequency-series object with the given
0027 % (x,y)-data. The sample rate is then set as
0028 % 2*x(end).
0029 % fsd = fsdata(y,fs) - creates a frequency-series object with the given
0030 % y-data and sample rate. The frequency
0031 % vector is grown assuming the first y
0032 % sample corresponds to 0Hz and the last
0033 % sample corresponds to the Nyquist
0034 % frequency.
0035 % fsd = fsdata(x,y,fs) - creates a frequency-series object with the given
0036 % x,y-data and sample rate.
0037 %
0038 % HISTORY: 30-01-2007 Hewitson
0039 % Creation
0040 %
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042
0043 VERSION = '$Id: fsdata.m,v 1.18 2007/11/30 19:36:04 ingo Exp $';
0044
0045 %%%%% Is this a 'Params' call? %%%%%
0046 if nargin == 2 && isa(varargin{1}, 'fsdata') && ischar(varargin{2})
0047     if strcmp(varargin{2}, 'Params')
0048         fsd = plist();
0049         return
0050     elseif strcmp(varargin{2}, 'Version')
0051         fsd = VERSION;
0052         return
0053     end
0054 end
0055
0056 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% define fsdata properties %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0057
0058 function fs = init()
0059     fs.name = 'None';
0060     fs.x = [];
0061     fs.y = [];
0062     fs.enbw = [];
0063     fs.navs = [];
0064     fs.fs = 0;
0065     fs.xunits = 'Hz';
0066     fs.yunits = '';
0067     fs.version = VERSION;
0068     fs.created = time;
0069     fs = class(fs, 'fsdata');
0070 end
0071
0072 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Create fsdata object %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0073
0074 %%%%%%%%% fsd = fsdata() %%%%%%%%%
0075 % create default fsdata object
0076 if nargin == 0

```

```

0077
0078     fsd = init();
0079
0080 elseif nargin == 1
0081     %%%%%%%%% Create from XML fragment %%%%%%%%%
0082     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0083         fsd = fromxml(varargin{1});
0084     %%%%%%%%% From File %%%%%%%%%
0085     elseif ischar(varargin{1})
0086
0087         filename = varargin{1};
0088         [path, name, ext, vers] = fileparts(filename);
0089         switch ext
0090             case '.mat'
0091                 fsd = load(filename);
0092                 fsd = fsd.a;
0093             case '.xml'
0094                 fsd = xmlparse(fsdata, filename);
0095             otherwise
0096                 error('### Unknown file type. ');
0097         end
0098     %%%%%%%%% fsd = fsdata(fsdata) %%%%%%%%%
0099     elseif isa(varargin{1}, 'fsdata')
0100
0101         fsd = varargin{1};
0102
0103     %%%%%%%%% fsd = fsdata(struct) %%%%%%%%%
0104     elseif isstruct(varargin{1})
0105
0106         fsd = init();
0107
0108         fields = fieldnames(varargin{1});
0109         for ii = 1:length(fields)
0110             field = fields{ii};
0111
0112             %%% created -> time-object
0113             if strcmp(field, 'created')
0114                 created = varargin{1}.created;
0115                 if isstruct(created)
0116                     created = time(created);
0117                 end
0118                 fsd.created = created;
0119             %%% All other
0120             else
0121                 try
0122                     fsd.(field) = varargin{1}.(field);
0123                 catch
0124                     error('### The field '%s' in the struct is not a fsdata property.',
0125                         field)
0126                 end
0127             end
0128         end
0129     end
0130 end

```

```

0128
0129 %%%%%%%%%% fsd = fsdata(y_vector) %%%%%%%%%%
0130 else
0131     fsd    = init();
0132     fsd.fs = 1;
0133     fsd.y  = varargin{1};
0134
0135     fsd    = setFreq(fsd);
0136 end
0137
0138 % Unify the y-axis and y-axis
0139 if size(fsd.y,2) > size(fsd.y,1)
0140     fsd.y = [fsd.y].';
0141 end
0142 if size(fsd.x, 1) ~= size(fsd.y,1)
0143     fsd.x = [fsd.x].';
0144 end
0145
0146 %%%%%%%%%% fsd = fsdata(x_vector, y_vector) %%%%%%%%%%
0147 elseif nargin == 2
0148     %%%%%%%%%% From DATABASE
0149     if isa(varargin{1}, 'database')
0150         fsd = retrieve(varargin{1}, varargin{2:end});
0151     elseif length(varargin{1}) == length(varargin{2})
0152
0153         fsd    = init();
0154         fsd.x  = varargin{1};
0155         fsd.y  = varargin{2};
0156         fsd.fs = fsd.x(end)*2;
0157
0158         if size(fsd.y,2) > size(fsd.y,1)
0159             fsd.y = [fsd.y].';
0160         end
0161         fsd = reshapeF(fsd);
0162
0163 %%%%%%%%%% fsd = fsdata(y_vector, fs) %%%%%%%%%%
0164 else
0165     % check we have fs properly here
0166     if length(varargin{2}) > 1
0167         error(['### unknown constructor call. Either x and y should be the' ...
0168             'same length, or fs should be a single value.']);
0169     end
0170
0171     fsd = init();
0172     fsd.y    = varargin{1};
0173     fsd.fs   = varargin{2};
0174
0175     if size(fsd.y,2) > size(fsd.y,1)
0176         fsd.y = [fsd.y].';
0177     end
0178     fsd = setFreq(fsd);
0179 end

```

```
0180
0181
0182 %%%%%%%%%% fsd = fsdata(x_vector, y_vector, fs) %%%%%%%%%%
0183 elseif nargin == 3
0184
0185     fsd = init();
0186     fsd.x      = varargin{1};
0187     fsd.y      = varargin{2};
0188     fsd.fs     = varargin{3};
0189
0190     if size(fsd.y,2) > size(fsd.y,1)
0191         fsd.y = [fsd.y].';
0192     end
0193     fsd = reshapeF(fsd);
0194 else
0195     error('### Unknown number of constructor arguments.');
```

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

```
0001 function f = mkhighpass(f, pl)
0002 % MKHIGHPASS return a high pass filter miir(). A Butterworth filter is used.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MKHIGHPASS return a high pass filter miir().
0007 % A Butterworth filter is used.
0008 %
```

```

0009 % CALL:          f = mkhighpass(f, pl)
0010 %
0011 % VERSION: $Id: mkhighpass.m,v 1.5 2007/10/24 10:55:11 ingo Exp $
0012 %
0013 % HISTORY: 27-08-2002 M Hewitson
0014 %          Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 g      = find(pl, 'gain');
0019 fc     = find(pl, 'fc');
0020 fs     = find(pl, 'fs');
0021 order = find(pl, 'order');
0022
0023
0024 if(fc > fs/2)
0025     error('fc must be < fs/2');
0026 end
0027
0028 f.name      = 'std highpass';
0029 f.fs        = fs;
0030 f.ntaps     = order+1;
0031 [f.a, f.b] = butter(order, 2*fc/fs, 'high');
0032 f.gain      = g;
0033 f.histin    = zeros(1,f.ntaps-1);      % initialise input history
0034 f.histout   = zeros(1,f.ntaps-1);      % initialise output history
0035
0036 % END
0037

```


[Home](#) > [classes](#) > [@time](#) > string.m

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 1.2 2008/01/15 17:36:36 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

```
0001 function cmd = string(tt, varargin)
0002 % STRING writes a command string that can be used to recreate the input time
```

```

object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0007 %                input time object.
0008 %
0009 % CALL:          cmd = string(pole);
0010 %
0011 % VERSION:       $Id: string.m,v 1.2 2008/01/15 17:36:36 ingo Exp $
0012 %
0013 % HISTORY:       07-08-2007 Diepholz
0014 %                Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: string.m,v 1.2 2008/01/15 17:36:36 ingo Exp $';
0019
0020 % Check if this is a call for parameters or for the cvs-version number
0021 if nargin == 2
0022     if isa(tt, 'time') && ischar(varargin{1})
0023         in = char(varargin{1});
0024         if strcmp(in, 'Params')
0025             cmd = plist();
0026             return
0027         elseif strcmp(in, 'Version')
0028             cmd = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 cmd = '[';
0035 for jj=1:length(tt)
0036
0037     pl = plist();
0038     pl = append(pl, 'timeformat',      char(tt(jj).timeformat));
0039     pl = append(pl, 'timezone',        char(tt(jj).timezone.getID));
0040     pl = append(pl, 'utc_epoch_milli', tt(jj).utc_epoch_milli);
0041
0042     cmd = [cmd ' time(' string(pl) ')'];
0043
0044 end
0045 cmd = [cmd ']';
0046
0047 % END

```

[Home](#) > [classes](#) > [@pzmodel](#) > [private](#) > Contents.m

Contents

PURPOSE

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: pzmodel/private %%%%%%%%%%
```

SYNOPSIS

This is a script file.

DESCRIPTION

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: pzmodel/private %%%%%%%%%%

    <a href="matlab:help pzmodel/private/abccascade">pzmodel/private/abccascade</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 22-Jan-2008 10:39:13 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.

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

GCBSH gets the handles for the currently selected blocks.

Usage: >> gcbsh
       >> h = gcbsh

M Hewitson 29-08-07

$Id: gcbsh.m,v 1.1 2007/08/29 06:30:16 hewitson Exp $

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

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = gcbsh()
0002
0003 % GCBSH gets the handles for the currently selected blocks.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % GCBSH gets the handles for the currently selected blocks.
0008 %
```

```

0009 % Usage: >> gcbsh
0010 %           >> h = gcbsh
0011 %
0012 % M Hewitson 29-08-07
0013 %
0014 % $Id: gcbsh.m,v 1.1 2007/08/29 06:30:16 hewitson Exp $
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 h = [];
0019
0020 blks = find_system(gcs, 'Type', 'block');
0021 hs    = get_param(blks, 'Handle');
0022
0023 % get selected blocks
0024 for j=1:length(hs)
0025     if strcmp(get(hs{j}, 'Selected'), 'on')
0026         h = [h hs{j}];
0027     end
0028 end
0029
0030 if nargout == 0
0031     disp(h.')
0032 elseif nargout == 1
0033     varargout{1} = h.';
0034 else
0035     error('### Too many output arguments')
0036 end

```

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

```
0001 function f = mklowpass(f, pl)
0002 % MKLOWPASS return a low pass filter mfir().
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MKLOWPASS return a low pass filter mfir().
0007 %
0008 % CALL:          f = mklowpass(f, pl)
0009 %
```

```

0010 % VERSION: $Id: mklowpass.m,v 1.4 2007/10/24 10:55:11 ingo Exp $
0011 %
0012 % HISTORY: 27-08-2002 M Hewitson
0013 %           Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 g      = find(pl, 'gain');
0018 fc     = find(pl, 'fc');
0019 fs     = find(pl, 'fs');
0020 order  = find(pl, 'order');
0021 win    = find(pl, 'Win');
0022
0023 if(fc(1) > fs/2)
0024     error('fc must be < fs/2');
0025 end
0026
0027 f.name    = 'std lowpass';
0028 f.fs      = fs;
0029 f.ntaps   = order+1;
0030 f.a       = fir1(order, 2*fc/fs, 'low', win.win);
0031 f.gd      = (f.ntaps)/2;
0032 f.gain    = g;
0033 f.histout = zeros(1,f.ntaps-1); % initialise output history
0034
0035 % END
0036

```

[Home](#) > [classes](#) > [@zero](#) > string.m

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 1.5 2008/01/03 15:33:49 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

```
0001 function cmd = string(ps, varargin)
0002 % STRING writes a command string that can be used to recreate the input zero
0003 % object.
```



```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0007 %               input zero object.
0008 %
0009 % CALL:         cmd = string (zero);
0010 %
0011 % VERSION:      $Id: string.m,v 1.5 2008/01/03 15:33:49 ingo Exp $
0012 %
0013 % HISTORY:      02-04-2007 Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: string.m,v 1.5 2008/01/03 15:33:49 ingo Exp $';
0019
0020 %%%% 'Params' && 'Version' Call %%%%
0021 if nargin == 2
0022     if isa(ps, 'zero') && ischar(varargin{1})
0023         in = varargin{1};
0024         if strcmp(in, 'Params')
0025             cmd = plist();
0026             return
0027         elseif strcmp(in, 'Version')
0028             cmd = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 cmd = '[';
0035 for j=1:length(ps)
0036     p = ps(j);
0037
0038     pl = plist();
0039     fields = fieldnames(p);
0040     for ii = 1:length(fields)
0041         field = fields{ii};
0042         if strcmp(field, 'ri')
0043             % do not add this field to the parametr list
0044         else
0045             pl = append(pl, field, p.(field));
0046         end
0047     end
0048
0049     cmd = [cmd ' zero(' string(pl) ')'];
0050 end
0051 cmd = [cmd ']';
0052
0053 % END

```


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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@miir](#) > [private](#) > mklowpass.m

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

```
0001 function f = mklowpass(f, pl)
0002 % MKLOWPASS return a low pass filter miir(). A Butterworth filter is used.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MKLOWPASS return a low pass filter miir().
0007 %           A Butterworth filter is used.
0008 %
```

```

0009 % CALL:          f = mklowpass(f, pl)
0010 %
0011 % VERSION: $Id: mklowpass.m,v 1.5 2007/10/24 10:55:11 ingo Exp $
0012 %
0013 % HISTORY: 27-08-2002 M Hewitson
0014 %          Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 g      = find(pl, 'gain');
0019 fc     = find(pl, 'fc');
0020 fs     = find(pl, 'fs');
0021 order = find(pl, 'order');
0022
0023 if(fc > fs/2)
0024     error('fc must be < fs/2');
0025 end
0026
0027 f.name      = 'std lowpass';
0028 f.fs        = fs;
0029 f.ntaps     = order+1;
0030 [f.a, f.b] = butter(order, 2*fc/fs);
0031 f.gain      = g;
0032 f.histin    = zeros(1,f.ntaps-1);    % initialise input history
0033 f.histout   = zeros(1,f.ntaps-1);    % initialise output history
0034
0035 % END
0036

```

[Home](#) > [m](#) > [helper](#) > strpad.m

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

```
0001 function so = strpad(s, N)
0002 % STRPAD Pads a string with blank spaces until it is N characters long.
```

```

0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: STRPAD Pads a string with blank spaces until it is N characters
0007 %                long. If s is already N characters long (or longer) then the
0008 %                string is truncated.
0009 %
0010 % CALL:          so = strpad('Pads this string to 30 characters', 30)
0011 %
0012 % INPUTS:        s - string
0013 %                N - length of the string
0014 %
0015 % OUTPUTS:       so - the padded string
0016 %
0017 % VERSION: $Id: strpad.m,v 1.2 2007/07/13 12:17:39 ingo Exp $
0018 %
0019 % HISTORY: 27-04-2007 M Hewitson
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 while length(s) < N
0025     s = [s ' '];
0026 end
0027
0028 so = s(1:N);
0029
0030
0031 % END

```


[Home](#) > [classes](#) > [@specwin](#) > [private](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@time](#) > Contents.m

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>       - save 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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@cdata](#) > get.m

get

PURPOSE

GET get cdata properties.

SYNOPSIS

function val = get(c, propName)

DESCRIPTION

GET get cdata properties.

%%%

DESCRIPTION: GET get cdata 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 1.5 2007/11/13 16:03:20 ingo Exp \$

HISTORY: 30-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a cdata object into a string.

This function is called by:

- [char](#) CHAR convert a cdata object into a string.
- [display](#) DISPLAY implement terminal display for cdata object.

- **xmladd** XMLADD add a cdata object to the given node of the input XML DOM

SOURCE CODE

```

0001 function val = get(c, propName)
0002 % GET get cdata properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get cdata properties.
0007 %
0008 % CALL:      name      = get(c, 'name');
0009 %           created    = get(c, 'created');
0010 %           version    = get(c, 'version');
0011 %           y          = get(c, 'y');
0012 %           y          = get(c, 'vals');
0013 %           x          = get(c, 'x');
0014 %           x          = get(c, 'tags');
0015 %           xunit      = get(c, 'xunits');
0016 %           yunit      = get(c, 'yunits');
0017 %
0018 % VERSION: $Id: get.m,v 1.5 2007/11/13 16:03:20 ingo Exp $
0019 %
0020 % HISTORY: 30-01-07 M Hewitson
0021 %           Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: get.m,v 1.5 2007/11/13 16:03:20 ingo Exp $';
0026
0027 % Check if this is a call for parameters
0028 if nargin == 2
0029     if isa(c, 'cdata') && ischar(propName)
0030         in = char(propName);
0031         if strcmp(in, 'Params')
0032             val = plist;
0033             return
0034         elseif strcmp(in, 'Version')
0035             val = VERSION;
0036             return
0037         end
0038     end
0039 end
0040
0041 switch propName
0042     case 'name'
0043         val = c.name;
0044     case 'created'
0045         val = c.created;
0046     case 'version'
0047         val = c.version;

```

```
0048     case 'vals'
0049         val = c.y;
0050     case 'y'
0051         val = c.y;
0052     case 'x'
0053         val = c.x;
0054     case 'tags'
0055         val = c.x;
0056     case 'xunits'
0057         val = c.xunits;
0058     case 'yunits'
0059         val = c.yunits;
0060     otherwise
0061         error([propName, ' is not a valid cdata property.']);
0062 end
```


[Home](#) > [classes](#) > [@time](#) > [private](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@fsdata](#) > get.m

get

PURPOSE

GET get a fsdata property.

SYNOPSIS

function val = get(fsd, propName)

DESCRIPTION

GET get a fsdata property.

%%%

DESCRIPTION: GET get fsdata properties.

```
CALL:      name      = get(fsd, 'name');
          fs         = get(fsd, 'fs');
          x          = get(fsd, 'x');
          enbw       = get(fsd, 'enbw');
          navs       = get(fsd, 'navs');
          y          = get(fsd, 'y');
          xunits     = get(fsd, 'xunits');
          yunits     = get(fsd, 'yunits');
          created    = get(fsd, 'created');
          version    = get(fsd, 'version');
```

VERSION: \$Id: get.m,v 1.10 2007/11/13 16:03:20 ingo Exp \$

HISTORY: 30-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a param object into a string.

This function is called by:

- [char](#) CHAR convert a param object into a string.

- **display** DISPLAY implement terminal display for fsdata object.
- **xmladd** XMLADD add a fsdata object to the given node of the input XML DOM

SOURCE CODE

```

0001 function val = get(fsd, propName)
0002 % GET get a fsdata property.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get fsdata properties.
0007 %
0008 % CALL:      name      = get(fsd, 'name');
0009 %           fs         = get(fsd, 'fs');
0010 %           x          = get(fsd, 'x');
0011 %           enbw       = get(fsd, 'enbw');
0012 %           navs       = get(fsd, 'navs');
0013 %           y          = get(fsd, 'y');
0014 %           xunits     = get(fsd, 'xunits');
0015 %           yunits     = get(fsd, 'yunits');
0016 %           created    = get(fsd, 'created');
0017 %           version    = get(fsd, 'version');
0018 %
0019 % VERSION: $Id: get.m,v 1.10 2007/11/13 16:03:20 ingo Exp $
0020 %
0021 % HISTORY: 30-01-07 M Hewitson
0022 %           Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025 %
0026 VERSION = '$Id: get.m,v 1.10 2007/11/13 16:03:20 ingo Exp $';
0027
0028 % Check if this is a call for parameters
0029 if nargin == 2
0030     if isa(fsd, 'fsdata') && ischar(propName)
0031         in = char(propName);
0032         if strcmp(in, 'Params')
0033             val = plist;
0034             return
0035         elseif strcmp(in, 'Version')
0036             val = VERSION;
0037             return
0038         end
0039     end
0040 end
0041
0042 switch propName
0043     case 'name'
0044         val = fsd.name;
0045     case 'created'
0046         val = fsd.created;

```

```
0047     case 'version'
0048         val = fsd.version;
0049     case 'fs'
0050         val = fsd.fs;
0051     case 'x'
0052         val = fsd.x;
0053     case 'enbw'
0054         val = fsd.enbw;
0055     case 'navs'
0056         val = fsd.navs;
0057     case 'y'
0058         val = fsd.y;
0059     case 'xunits'
0060         val = fsd.xunits;
0061     case 'yunits'
0062         val = fsd.yunits;
0063     otherwise
0064         error([propName, ' is not a valid fsdata property.']);
0065 end
```

[Home](#) > [classes](#) > [@cdata](#) > subsasgn.m

subsasgn

PURPOSE

SUBSASGN define index assignment for vectors in tsdata.

SYNOPSIS

function ts = subsasgn(ts, index, val)

DESCRIPTION

SUBSASGN define index assignment for vectors in tsdata.

M Hewitson 30-01-07

\$Id: subsasgn.m,v 1.3 2007/11/23 12:11:08 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a cdata object into a string.

This function is called by:

SOURCE CODE

```
0001 function ts = subsasgn(ts, index, val)
0002
0003 % SUBSASGN define index assignment for vectors in tsdata.
0004 %
0005 % M Hewitson 30-01-07
0006 %
0007 % $Id: subsasgn.m,v 1.3 2007/11/23 12:11:08 ingo Exp $
0008
```

```
0009 VERSION = '$Id: subsasgn.m,v 1.3 2007/11/23 12:11:08 ingo Exp $';
0010
0011 % Check if this is a call for parameters
0012 if nargin == 2
0013     if isa(ts, 'cdata') && ischar(index)
0014         in = char(index);
0015         if strcmp(in, 'Params')
0016             ts = plist();
0017             return
0018         elseif strcmp(in, 'Version')
0019             ts = VERSION;
0020             return
0021         end
0022     end
0023 end
0024
0025 % END
```

Generated on Tue 22-Jan-2008 10:39:13 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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@history](#) > get.m

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 1.6 2007/12/04 18:16:11 ingo Exp \$

HISTORY: 30-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a param object into a string.

This function is called by:

- [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.
- **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.
- **xmladd** XMLADD add an history object to the given node of the input XML DOM

SOURCE CODE

```

0001 function val = get(h, propName)
0002 % GET get history object properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get history object properties.
0007 %
0008 % CALL: name      = get(h, 'name');
0009 %          ver      = get(h, 'version');
0010 %          pl       = get(h, 'plist');
0011 %          inhist   = get(h, 'inhist');
0012 %          invars   = get(h, 'invars');
0013 %          n        = get(h, 'n');
0014 %          pn       = get(h, 'pn');
0015 %          created  = get(h, 'created');
0016 %          consver  = get(h, 'consver');
0017 %
0018 % VERSION: $Id: get.m,v 1.6 2007/12/04 18:16:11 ingo Exp $
0019 %
0020 % HISTORY: 30-01-07 M Hewitson
0021 %          Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: get.m,v 1.6 2007/12/04 18:16:11 ingo Exp $';
0026
0027 % Check if this is a call for parameters
0028 if nargin == 2
0029     if isa(h, 'history') && ischar(propName)
0030         in = char(propName);
0031         if strcmp(in, 'Params')
0032             val = plist;
0033             return
0034         elseif strcmp(in, 'Version')
0035             val = VERSION;
0036             return
0037         end
0038     end
0039 end
0040
0041 switch propName
0042     case 'name'

```

```
0043     val = h.name;
0044     case 'version'
0045         val = h.version;
0046     case 'plist'
0047         val = h.plist;
0048     case 'inhists'
0049         val = h.inhists;
0050     case 'invars'
0051         val = h.invars;
0052     case 'n'
0053         val = h.n;
0054     case 'pn'
0055         val = h.pn;
0056     case 'created'
0057         val = h.created;
0058     case 'consver'
0059         val = h.consver;
0060     otherwise
0061         error([propName, ' is not a valid history property.']);
0062 end
0063
```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > muitable.m

mutable

PURPOSE

UITABLE creates a two dimensional graphic uitable component in a figure window.

SYNOPSIS

function [table, container] = mutable(varargin)

DESCRIPTION

UITABLE creates a two dimensional graphic uitable component in a figure window. **UITABLE** creates a 1x1 uitable object using default property values in a figure window.

UITABLE(numrows,numcolumns) creates a uitable object with specified number of rows and columns.

UITABLE(data,columnNames) creates a uitable object with the specified data and columnNames. Data can be a cell array or a vector and columnNames should be cell arrays.

UITABLE('PropertyName1',value1,'PropertyName2',value2,...) creates a uitable object with specified property values. MATLAB uses default property values for any property not explicitly set. The properties that user can set are: ColumnNames, Data, GridColor, NumColumns, NumRows, Position, ColumnWidth and RowHeight.

UITABLE(figurehandle, ...) creates a uitable object in the figure window specified by the figure handle.

HANDLE = **UITABLE**(...) creates a uitable object and returns its handle.

Properties:

ColumnNames:	Cell array of strings for column names.
Data:	Cell array of values to be displayed in the table.
GridColor:	string, RGB vector.
NumColumns:	int specifying number of columns.
NumRows:	int specifying number of rows.
Parent:	Handle to figure or uipanel. If not specified, it is gcf.
Position:	4 element vector specifying the position.
ColumnWidth:	int specifying the width of columns.
RowHeight:	int specifying the height of columns.

Enabled: Boolean specifying if a column is enabled.

Editable: Boolean specifying if a column is editable.
Units: String - pixels/normalized/inches/points/centimeters.
Visible: Boolean specifying if table is visible.
DataChangedCallback - Callback function name or handle.

Examples:

```
t = uitable(3, 2);
```

Creates a 3x2 empty uitable object in a figure window.

```
f = figure;
t = uitable(f, rand(5), {'A', 'B', 'C', 'D', 'E'});
```

Creates a 5x5 uitable object in a figure window with the specified data and the column names.

```
data = rand(3);
colnames = {'X-Data', 'Y-Data', 'Z-Data'};
t = uitable(data, colnames, 'Position', [20 20 250 100]);
```

Creates a uitable object with the specified data and column names and the specified Position.

See also [AWTCREATE](#), [AWTINVOKE](#), [JAVACOMPONENT](#), [UITREE](#), [UITREENODE](#)

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.
0004 %      UITABLE creates a 1x1 uitable object using default property values in
0005 %      a figure window.
0006 %
0007 %      UITABLE(numrows,numcolumns) creates a uitable object with specified
0008 %      number of rows and columns.

```

```

0009 %
0010 %    Uitable(data,columnNames) creates a uitable object with the specified
0011 %    data and columnNames. Data can be a cell array or a vector and
0012 %    columnNames should be cell arrays.
0013 %
0014 %    Uitable('PropertyName1',value1,'PropertyName2',value2,...) creates a
0015 %    uitable object with specified property values. MATLAB uses default
0016 %    property values for any property not explicitly set. The properties
0017 %    that user can set are: ColumnNames, Data, GridColor, NumColumns,
0018 %    NumRows, Position, ColumnWidth and RowHeight.
0019 %
0020 %    Uitable(figurehandle, ...) creates a uitable object in the figure
0021 %    window specified by the figure handle.
0022 %
0023 %    HANDLE = Uitable(...) creates a uitable object and returns its handle.
0024 %
0025 %    Properties:
0026 %
0027 %    ColumnNames: Cell array of strings for column names.
0028 %    Data:        Cell array of values to be displayed in the table.
0029 %    GridColor:   string, RGB vector.
0030 %    NumColumns:  int specifying number of columns.
0031 %    NumRows:     int specifying number of rows.
0032 %    Parent:      Handle to figure or uipanel. If not specified, it is gcf.
0033 %    Position:    4 element vector specifying the position.
0034 %    ColumnWidth: int specifying the width of columns.
0035 %    RowHeight:   int specifying the height of columns.
0036 %
0037 %    Enabled:      Boolean specifying if a column is enabled.
0038 %    Editable:     Boolean specifying if a column is editable.
0039 %    Units:        String - pixels/normalized/inches/points/centimeters.
0040 %    Visible:      Boolean specifying if table is visible.
0041 %    DataChangedCallback - Callback function name or handle.
0042 %
0043 %
0044 %    Examples:
0045 %
0046 %    t = uitable(3, 2);
0047 %
0048 %    Creates a 3x2 empty uitable object in a figure window.
0049 %
0050 %    f = figure;
0051 %    t = uitable(f, rand(5), {'A', 'B', 'C', 'D', 'E'});
0052 %
0053 %    Creates a 5x5 uitable object in a figure window with the specified
0054 %    data and the column names.
0055 %
0056 %    data = rand(3);
0057 %    colnames = {'X-Data', 'Y-Data', 'Z-Data'};
0058 %    t = uitable(data, colnames,'Position',[20 20 250 100]);
0059 %
0060 %    Creates a uitable object with the specified data and column names and
0061 %    the specified Position.

```

```

0062 %
0063 %     See also AWTCREATE, AWTINVOKE, JAVACOMPONENT, UITREE, UITREENODE
0064
0065 %     Copyright 2002-2006 The MathWorks, Inc.
0066 %     $Revision: 1.1 $   $Date: 2007/09/15 04:56:44 $
0067
0068 %     Release: R14. This feature will not work in previous versions of MATLAB.
0069
0070 % Setup and P-V parsing
0071
0072 error(javachk('awt'));
0073 error(nargoutchk(0,2,nargout));
0074
0075 parent = [];
0076 numargs = nargin;
0077
0078 datastatus=false; columnstatus=false;
0079 rownum = 1; colnum = 1; % Default to a 1x1 table.
0080 position = [20 20 200 200];
0081 combo_box_found = false;
0082 check_box_found = false;
0083
0084 import com.mathworks.hg.peer.UitablePeer;
0085
0086 if (numargs > 0 && isscalar(varargin{1}) && ishandle(varargin{1}) && ...
0087     isa(handle(varargin{1}), 'figure'))
0088     parent = varargin{1};
0089     varargin = varargin(2:end);
0090     numargs = numargs - 1;
0091 end
0092
0093 if (numargs > 0 && isscalar(varargin{1}) && ishandle(varargin{1}))
0094     if ~isa(varargin{1}, 'javax.swing.table.DefaultTableModel')
0095         error('MATLAB:uitable:UnrecognizedParameter', ['Unrecognized parameter:
0096 ', varargin{1}]);
0097     end
0098     data_model = varargin{1};
0099     varargin = varargin(2:end);
0100     numargs = numargs - 1;
0101
0102 elseif ((numargs > 1) && isnumeric(varargin{1}) && isnumeric(varargin{2}))
0103     if(isnumeric(varargin{1}) && isnumeric(varargin{2}))
0104         rownum = varargin{1};
0105         colnum = varargin{2};
0106
0107         varargin = varargin(3:end);
0108         numargs = numargs-2;
0109     else
0110         error('MATLAB:uitable:InputMustBeScalar', 'When using UITABLE numRows
0111 and numcols have to be numeric scalars.')
0112     end
0113
0114 elseif ((numargs > 1) && isequal(size(varargin{2},1), 1) && iscell(varargin{2}))

```

```

0113     if (size(varargin{1},2) == size(varargin{2},2))
0114         if (isnumeric(varargin{1}))
0115             varargin{1} = num2cell(varargin{1});
0116         end
0117     else
0118         error('MATLAB:uitable:MustMatchInfo', 'Number of column names must
match number of columns in data');
0119     end
0120     data = varargin{1};      datastatus      = true;
0121     coln = varargin{1+1};    columnstatus    = true;
0122
0123     varargin = varargin(3:end);
0124     numargs = numargs-2;
0125 end
0126
0127 for i = 1:2:numargs-1
0128     if (~ischar(varargin{i}))
0129         error('MATLAB:uitable:UnrecognizedParameter', ['Unrecognized parameter:
', varargin{i}]);
0130     end
0131     switch lower(varargin{i})
0132     case 'data'
0133         if (isnumeric(varargin{i+1}))
0134             varargin{i+1} = num2cell(varargin{i+1});
0135         end
0136         data      = varargin{i+1};
0137         datastatus = true;
0138
0139     case 'columnnames'
0140         if(iscell(varargin{i+1}))
0141             coln      = varargin{i+1};
0142             columnstatus = true;
0143         else
0144             error('MATLAB:uitable:InvalidCellArray', 'When using Uitable
Column data should be 1xn cell array')
0145         end
0146
0147     case 'numrows'
0148         if (isnumeric(varargin{i+1}))
0149             rownum = varargin{i+1};
0150         else
0151             error('MATLAB:uitable:NumrowsMustBeScalar', 'numrows has to be
a scalar')
0152         end
0153
0154     case 'numcolumns'
0155         if (isnumeric(varargin{i+1}))
0156             colnum = varargin{i+1};
0157         else
0158             error('MATLAB:uitable:NumcolumnsMustBeScalar', 'numcolumns has
to be a scalar')
0159         end
0160

```



```

0161         case 'gridcolor'
0162             if (ischar(varargin{i+1}))
0163                 gridcolor = varargin{i+1};
0164             else if (isnumeric(varargin{i+1}) && (numel(varargin{i+1}) == 3))
0165                 gridcolor = varargin{i+1};
0166             else
0167                 error('MATLAB:uitable:InvalidString', 'gridcolor has to be
a valid string')
0168             end
0169         end
0170
0171         case 'rowheight'
0172             if (isnumeric(varargin{i+1}))
0173                 rowheight = varargin{i+1};
0174             else
0175                 error('MATLAB:uitable:RowheightMustBeScalar', 'rowheight has to
be a scalar')
0176             end
0177
0178         case 'parent'
0179             if ishandle(varargin{i+1})
0180                 parent = varargin{i+1};
0181             else
0182                 error('MATLAB:uitable:InvalidParent', 'parent must be a valid
handle')
0183             end
0184
0185         case 'position'
0186             if (isnumeric(varargin{i+1}))
0187                 position = varargin{i+1};
0188             else
0189                 error('MATLAB:uitable:InvalidPosition', 'position has to be a
1x4 numeric array')
0190             end
0191
0192         case 'columnwidth'
0193             if (isnumeric(varargin{i+1}))
0194                 columnwidth = varargin{i+1};
0195             else
0196                 error('MATLAB:uitable:ColumnwidthMustBeScalar', 'columnwidth
has to be a scalar')
0197             end
0198         otherwise
0199             error('MATLAB:uitable:UnrecognizedParameter', ['Unrecognized
parameter: ', varargin{i}]);
0200     end
0201 end
0202
0203 % ---combo/check box detection--- %
0204 if (datastatus)
0205     if (iscell(data))
0206         rownum = size(data,1);
0207         colnum = size(data,2);

```

```

0208         combo_count =0;
0209         check_count = 0;
0210         combo_box_data = num2cell(zeros(1, colnum));
0211         combo_box_column = zeros(1, colnum);
0212         check_box_column = zeros(1, colnum);
0213         for j = 1:rownum
0214             for k = 1:colnum
0215                 if (iscell(data{j,k}))
0216                     combo_box_found = true;
0217                     combo_count = combo_count + 1;
0218                     combo_box_data{combo_count} = data{j,k};
0219                     combo_box_column(combo_count) = k;
0220                     dc = data{j,k};
0221                     data{j,k} = dc{1};
0222                 else
0223                     if(islogical(data{j,k}))
0224                         check_box_found = true;
0225                         check_count = check_count + 1;
0226                         check_box_column(check_count) = k;
0227                     end
0228                 end
0229             end
0230         end
0231     end
0232 end
0233
0234 %%%%%%%%%%%%%%%
0235 % Check the validity of the parent and/or create a figure.
0236 if isempty(parent)
0237     parent =(gcf); % Get the current figure. Create one if not available
0238 end
0239
0240 if ( columnstatus && datastatus )
0241     if(size(data,2) ~= size(coln,2))
0242         error('MATLAB:NeedSameNumberColumns', 'Number of columns in both Data
and ColumnNames should match');
0243     end
0244 elseif ( ~columnstatus && datastatus )
0245     for i=1:size(data,2)
0246         coln{i} = num2str(i);
0247     end
0248     columnstatus = true;
0249 elseif ( columnstatus && ~datastatus)
0250     error('MATLAB:uitable:NoDataProvided', 'No Data provided along with
ColumnNames');
0251 end
0252
0253 if (~exist('data_model', 'var'))
0254     data_model = javax.swing.table.DefaultTableModel;
0255 end
0256 if exist('rownum', 'var')
0257     data_model.setRowCount(rownum);
0258 end

```

```

0259 if exist('colnum', 'var')
0260     data_model.setColumnCount(colnum);
0261 end
0262
0263 table_h= UitablePeer(data_model);
0264
0265 % We should have valid data and column names here.
0266 if (datastatus), table_h.setData(data); end;
0267 if (columnstatus), table_h.setColumnNames(coln); end;
0268
0269 if (combo_box_found),
0270     for i=1:combo_count
0271         table_h.setComboBoxEditor(combo_box_data(i), combo_box_column(i));
0272     end
0273 end
0274 if (check_box_found),
0275     for i = 1: check_count
0276         table_h.setCheckBoxEditor(check_box_column(i));
0277     end
0278 end
0279
0280 % pass the specified parent and let javacomponent decide its validity.
0281 [obj, container] = javacomponent(table_h, position, parent);
0282 % javacomponent returns a UDD handle for the java component passed in.
0283 table = obj;
0284
0285 % Have to do a drawnow here to make the properties stick. Try to restrict
0286 % the drawnow call to only when it is absolutely required.
0287 flushed = false;
0288 if exist('gridcolor', 'var')
0289     pause(.1); drawnow;
0290     flushed = true;
0291     table_h.setGridColor(gridcolor);
0292 end
0293 if exist('rowheight', 'var')
0294     if (~flushed)
0295         drawnow;
0296     end
0297     table_h.setRowHeight(rowheight);
0298 end
0299 if exist('columnwidth', 'var')
0300     table_h.setColumnWidth(columnwidth);
0301 end;
0302
0303 % Add a predestroy listener so we can call cleanup on the table.
0304 % addlistener(table, 'ObjectBeingDestroyed', {@componentDelete});
0305
0306 function componentDelete(src, evd) %#ok
0307 % Clean up the table here so it disengages all its internal listeners.
0308 src.cleanup;

```


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 1.3 2007/11/23 12:11:08 ingo 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for fsdata properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for fsdata
```

```

0007 %                properties.
0008 %
0009 % EXAMPLES:      All possible assignment are possible.
0010 %
0011 % VERSION:        $Id: subsasgn.m,v 1.3 2007/11/23 12:11:08 ingo Exp $
0012 %
0013 % HISTORY:        31-01-07 M Hewitson
0014 %                  Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 VERSION = '$Id: subsasgn.m,v 1.3 2007/11/23 12:11:08 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(A, 'fsdata') && ischar(S)
0022         in = char(S);
0023         if strcmp(in, 'Params')
0024             A = plist;
0025             return
0026         elseif strcmp(in, 'Version')
0027             A = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 A = builtin ('subsasgn', A, S, B);
0034

```

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:

get

PURPOSE

GET get mfir properties.

SYNOPSIS

function val = get(varargin)

DESCRIPTION

GET get mfir properties.

%%%

DESCRIPTION: GET get mfir properties.

CALL: get(miir)
 val = get(miir, property);
 name = get(miir, 'name');
 fs = get(miir, 'fs');
 ntaps = get(miir, 'ntaps');
 a = get(miir, 'a');
 gd = get(miir, 'gd');
 gain = get(miir, 'gain');
 history = get(miir, 'histout');
 infile = get(miir, 'infile');
 plist = get(miir, 'plist');
 created = get(miir, 'created');
 version = get(miir, 'version');

VERSION: \$Id: get.m,v 1.3 2007/12/06 17:09:15 ingo Exp \$

HISTORY: 30-01-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a mfir object into a string.

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.
- **xmladd** XMLADD add an MFIR object to the given node of the input XML DOM

SOURCE CODE

```

0001 function val = get(varargin)
0002 % GET get mfir properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get mfir properties.
0007 %
0008 % CALL:          get(miir)
0009 %              val      = get(miir, property);
0010 %              name     = get(miir, 'name');
0011 %              fs       = get(miir, 'fs');
0012 %              ntaps    = get(miir, 'ntaps');
0013 %              a        = get(miir, 'a');
0014 %              gd       = get(miir, 'gd');
0015 %              gain     = get(miir, 'gain');
0016 %              history  = get(miir, 'histout');
0017 %              infile   = get(miir, 'infile');
0018 %              plist    = get(miir, 'plist');
0019 %              created  = get(miir, 'created');
0020 %              version  = get(miir, 'version');
0021 %
0022 % VERSION:       $Id: get.m,v 1.3 2007/12/06 17:09:15 ingo Exp $
0023 %
0024 % HISTORY:      30-01-2007 M Hewitson
0025 %               Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 VERSION = '$Id: get.m,v 1.3 2007/12/06 17:09:15 ingo Exp $';
0030
0031 % Check if this is a call for parameters
0032 if nargin == 2
0033     if isa(varargin{1}, 'mfir') && ischar(varargin{2})
0034         in = char(varargin{2});
0035         if strcmp(in, 'Params')
0036             val = plist;
0037             return
0038         elseif strcmp(in, 'Version')
0039             val = VERSION;
0040             return

```

```
0041     end
0042 end
0043 end
0044
0045 switch nargin
0046     case 1
0047         f = varargin{1};
0048         disp(f);
0049
0050     case 2
0051         f          = varargin{1};
0052         propName = varargin{2};
0053
0054         switch propName
0055             case 'name'
0056                 val = f.name;
0057             case 'fs'
0058                 val = f.fs;
0059             case 'ntaps'
0060                 val = f.ntaps;
0061             case 'a'
0062                 val = f.a;
0063             case 'gd'
0064                 val = f.gd;
0065             case 'gain'
0066                 val = f.gain;
0067             case 'histout'
0068                 val = f.histout;
0069             case 'infile'
0070                 val = f.infile;
0071             case 'plist'
0072                 val = f.plist;
0073             case 'created'
0074                 val = f.created;
0075             case 'version'
0076                 val = f.version;
0077             otherwise
0078                 error([propName, ' is not a valid mfir property.']);
0079         end
0080     otherwise
0081         error('### too many input arguments. ');
0082 end
```

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 1.2 2008/01/07 13:54:11 ingo 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for history properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for history
```

```

0007 %                properties.
0008 %
0009 % EXAMPLES:      All possible assignment are possible.
0010 %
0011 % VERSION:        $Id: subsasgn.m,v 1.2 2008/01/07 13:54:11 ingo Exp $
0012 %
0013 % HISTORY:        31-01-07 M Hewitson
0014 %                  Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 VERSION = '$Id: subsasgn.m,v 1.2 2008/01/07 13:54:11 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(A, 'history') && ischar(S)
0022         in = char(S);
0023         if strcmp(in, 'Params')
0024             A = plist;
0025             return
0026         elseif strcmp(in, 'Version')
0027             A = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 A = builtin ('subsasgn', A, S, B);
0034

```

[Home](#) > [classes](#) > [@timespan](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@miir](#) > get.m

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 1.5 2007/12/06 17:09:15 ingo Exp \$

HISTORY: 30-01-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a miir object into a string.

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.
- **xmladd** XMLADD add an MIIR object to the given node of the input XML DOM

SOURCE CODE

```

0001 function val = get(varargin)
0002 % GET get miir properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get miir properties.
0007 %
0008 % CALL:          get(miir)
0009 %              val      = get(miir, property);
0010 %              name     = get(miir, 'name');
0011 %              version  = get(miir, 'version');
0012 %              created  = get(miir, 'created');
0013 %              fs       = get(miir, 'fs');
0014 %              ntaps    = get(miir, 'ntaps');
0015 %              a        = get(miir, 'a');
0016 %              b        = get(miir, 'b');
0017 %              gain     = get(miir, 'gain');
0018 %              infile   = get(miir, 'infile');
0019 %              history  = get(miir, 'histin');
0020 %              history  = get(miir, 'histout');
0021 %              pl       = get(miir, 'plist');
0022 %
0023 % VERSION:       $Id: get.m,v 1.5 2007/12/06 17:09:15 ingo Exp $
0024 %
0025 % HISTORY:       30-01-2007 M Hewitson
0026 %                Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 VERSION = '$Id: get.m,v 1.5 2007/12/06 17:09:15 ingo Exp $';
0031
0032 % Check if this is a call for parameters
0033 if nargin == 2
0034     if isa(varargin{1}, 'miir') && ischar(varargin{2})
0035         in = char(varargin{2});
0036         if strcmp(in, 'Params')
0037             val = plist;
0038             return
0039         elseif strcmp(in, 'Version')
0040             val = VERSION;

```



```
0041     return
0042 end
0043 end
0044 end
0045
0046 switch nargin
0047     case 1
0048         f = varargin{1};
0049         disp(f);
0050
0051     case 2
0052         f = varargin{1};
0053         propName = varargin{2};
0054
0055         switch propName
0056             case 'name'
0057                 val = f.name;
0058             case 'fs'
0059                 val = f.fs;
0060             case 'ntaps'
0061                 val = f.ntaps;
0062             case 'a'
0063                 val = f.a;
0064             case 'b'
0065                 val = f.b;
0066             case 'gain'
0067                 val = f.gain;
0068             case 'histin'
0069                 val = f.histin;
0070             case 'histout'
0071                 val = f.histout;
0072             case 'infile'
0073                 val = f.infile;
0074             case 'created'
0075                 val = f.created;
0076             case 'version'
0077                 val = f.version;
0078             case 'plist'
0079                 val = f.plist;
0080             otherwise
0081                 error([propName, ' is not a valid miir property.']);
0082         end
0083     otherwise
0084         error('### too many input arguments. ');
0085 end
```

mux

PURPOSE

MUX concatonate params into a vector.

SYNOPSIS

function b = mux(varargin)

DESCRIPTION

MUX concatonate params into a vector.

%%%

DESCRIPTION: MUX concatonate params into a vector.

CALL: param_vec = mux(p1, p2);

VERSION: \$Id: mux.m,v 1.3 2007/12/11 17:27:09 ingo 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

```
0001 function b = mux(varargin)
0002 % MUX concatonate params into a vector.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MUX concatonate params into a vector.
0007 %
```

```

0008 % CALL:      param_vec = mux(p1, p2);
0009 %
0010 % VERSION:    $Id: mux.m,v 1.3 2007/12/11 17:27:09 ingo Exp $
0011 %
0012 % HISTORY:    27-03-2007 M Hewitson
0013 %              Creation
0014 %
0015 %%%%%%%%%%%
0016
0017 VERSION = '$Id: mux.m,v 1.3 2007/12/11 17:27:09 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(varargin{1}, 'param') && ischar(varargin{2})
0022         in = char(varargin{2});
0023         if strcmp(in, 'Params')
0024             b = plist();
0025             return
0026         elseif strcmp(in, 'Version')
0027             b = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 b = [];
0034 for j=1:nargin
0035     if isa(varargin{j}, 'param')
0036         b = [b varargin{j}];
0037     end
0038 end
0039

```

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 1.1 2008/01/07 13:46:02 ingo 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for mfir properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for mfir properties.
0007 %
```

```

0008 % EXAMPLES:      All possible assignment are possible.
0009 %
0010 % VERSION:        $Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $
0011 %
0012 % HISTORY:        31-01-07 M Hewitson
0013 %                  Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'mfir') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             A = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             A = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 A = builtin ('subsasgn', A, S, B);
0033

```

[Home](#) > [classes](#) > [@timespan](#) > [private](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@param](#) > get.m

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 1.4 2007/10/15 17:16:56 ingo Exp \$

HISTORY: 30-01-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a param object into a string.

This function is called by:

- [char](#) CHAR convert a param object into a string.
- [display](#) DISPLAY display a parameter

SOURCE CODE

```

0001 function val = get(p, propName)
0002 % GET get parameter properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get parameter properties.
0007 %
0008 % CALL:          name      = get(parameter, 'name');
0009 %               key       = get(parameter, 'key');
0010 %               val       = get(parameter, 'val');
0011 %               created   = get(parameter, 'created');
0012 %               version   = get(parameter, 'version');
0013 %
0014 % VERSION:       $Id: get.m,v 1.4 2007/10/15 17:16:56 ingo Exp $
0015 %
0016 % HISTORY:       30-01-2007 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: get.m,v 1.4 2007/10/15 17:16:56 ingo Exp $';
0022
0023 % Check if this is a call for parameters or for the cvs-version number
0024 if nargin == 2
0025     if isa(p, 'param') && ischar(propName)
0026         in = char(propName);
0027         if strcmp(in, 'Params')
0028             val = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             val = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 switch propName
0038     case 'name'
0039         val = p.name;
0040     case 'key'
0041         val = p.key;
0042     case 'val'
0043         val = p.val;
0044     case 'created'
0045         val = p.created;
0046     case 'version'
0047         val = p.version;
0048     otherwise
0049         error([propName, ' is not a valid param property.']);
0050 end

```


[Home](#) > [m](#) > [mysql](#) > mysql_connect.m

mysql_connect

PURPOSE

MYSQL_CONNECT connects to an AO repository and returns the connection object.

SYNOPSIS

function [conn, dbuser] = 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 1.7 2007/09/15 04:52:07 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_dbquery](#) LTPDA_DBQUERY query an AO repository database.

SOURCE CODE

```
0001 function [conn, dbuser] = mysql_connect(varargin)
0002 % MYSQL_CONNECT connects to an AO repository and returns the connection object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

0005 %
0006 % DESCRIPTION: MYSQL_CONNECT connects to an A0 repository and returns the
0007 %               connection object.
0008 %
0009 % CALL:         conn = mysql_connect(hostname, dbname)
0010 %
0011 % VERSION:      $Id: mysql_connect.m,v 1.7 2007/09/15 04:52:07 hewitson Exp $
0012 %
0013 % HISTORY:      24-05-2007 M Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%
0017
0018 if nargin < 1
0019     error('### Incorrect inputs')
0020 end
0021
0022 if nargin == 1
0023     dbhost = varargin{1};
0024     dbname = 'mysql';
0025 else
0026     dbhost = varargin{1};
0027     dbname = varargin{2};
0028 end
0029
0030 %% Prompt for username and password
0031
0032 % first try defaults
0033 dbuser = getappdata(0, 'ltpda_repo_user');
0034 dbpass = getappdata(0, 'ltpda_repo_pass');
0035
0036 if isempty(dbuser) || isempty(dbpass)
0037     [dbuser, dbpass] = logindlg('Title','LTPDA Repository Access');
0038
0039     if isempty(dbuser) || isempty(dbpass)
0040         warning('!!! Login process cancelled.');
```

conn = [];

```

0042     return
0043 end
0044 end
0045
0046 %% Database settings
0047
0048 dbdriver = getappdata(0, 'mysql_driver');
0049 dburl     = sprintf('jdbc:mysql://%s/%s',dbhost,dbname);
0050
0051 disp(sprintf('** Connecting to %s as %s...', dbhost, dbuser))
0052
0053 conn      = database(dbname,dbuser,dbpass,dbdriver,dburl);
0054
0055 disp('** Connection status:')
0056 disp(ping(conn))

```

0057

0058

0059 % END

Generated on Tue 22-Jan-2008 10:39:13 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 1.1 2008/01/07 13:46:02 ingo 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for miir properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for miir properties.
0007 %
```

```

0008 % EXAMPLES:      All possible assignment are possible.
0009 %
0010 % VERSION:        $Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'miir') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             A = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             A = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 A = builtin ('subsasgn', A, S, B);
0033

```

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.

[tsdata/xmladd](matlab:help tsdata/xmladd)

- add a tsdata

object to the given node of the input XML DOM

[tsdata/xmlparse](matlab:help tsdata/xmlparse)

- creates a

tsdata from a tsdata xml file.

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [mysql](#) > mysql_getMD5hash.m

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 [^](#)

```

0001 function h = mysql_getMD5hash(conn, id)
0002
0003 % MYSQL_GETMD5HASH gets the md5 hash string of a given object ID.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: MYSQL_GETMD5HASH gets the md5 hash string of a given object ID.
0008 %
0009 % CALL:            h = mysql_getMD5hash(conn, id)

```

```
0010 %
0011 % VERSION:      $Id: mysql_getMD5hash.m,v 1.1 2007/09/15 04:54:44 hewitson Exp $
0012 %
0013 % HISTORY:      24-05-2007 M Hewitson
0014 %                Creation
0015 %
0016 %%%%%%%%%%%%%%%
0017
0018 q      = sprintf('select hash from objs where id="%d"', id);
0019 curs = exec(conn, q);
0020 curs = fetch(curs);
0021 h = curs.Data{1};
0022 close(curs);
0023
```

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 1.4 2007/12/11 17:27:09 ingo 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for parameter properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for parameter properties.
0007 %
```

```

0008 % EXAMPLES:      All possible assignment are possible.
0009 %
0010 % VERSION:        $Id: subsasgn.m,v 1.4 2007/12/11 17:27:09 ingo Exp $
0011 %
0012 % HISTORY:        31-01-07 M Hewitson
0013 %                  Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsasgn.m,v 1.4 2007/12/11 17:27:09 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'param') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             A = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             A = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 A = builtin ('subsasgn', A, S, B);
0033

```

[Home](#) > [classes](#) > [@tsdata](#) > [private](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@pole](#) > get.m

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 1.5 2007/10/16 17:37:21 ingo Exp \$

HISTORY: 02-04-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a pole object into a string.

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.

- **xmladd** XMLADD add a pole object to the given node of the input XML DOM

SOURCE CODE

```

0001 function val = get(p, propName)
0002 % GET get pole properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get pole properties.
0007 %
0008 % CALL:          name      = get(p, 'name');
0009 %               f         = get(p, 'f');
0010 %               q         = get(p, 'q');
0011 %               ri        = get(p, 'ri');
0012 %               created    = get(p, 'created');
0013 %               version    = get(p, 'version');
0014 %
0015 % VERSION:       $Id: get.m,v 1.5 2007/10/16 17:37:21 ingo Exp $
0016 %
0017 % HISTORY:       02-04-07 M Hewitson
0018 %               Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: get.m,v 1.5 2007/10/16 17:37:21 ingo Exp $';
0023
0024 % Check if this is a call for parameters or for the cvs-version number
0025 if nargin == 2
0026     if isa(p, 'pole') && ischar(propName)
0027         in = char(propName);
0028         if strcmp(in, 'Params')
0029             val = plist();
0030             return
0031         elseif strcmp(in, 'Version')
0032             val = VERSION;
0033             return
0034         end
0035     end
0036 end
0037
0038 switch propName
0039     case 'name'
0040         val = p.name;
0041     case 'f'
0042         val = p.f;
0043     case 'q'
0044         val = p.q;
0045     case 'ri'
0046         val = p.ri;
0047     case 'created'

```

```
0048     val = p.created;  
0049     case 'version'  
0050         val = p.version;  
0051     otherwise  
0052         error([propName, ' is not a valid pole property.']);  
0053 end
```


[Home](#) > [m](#) > [mysql](#) > mysql_getMaxId.m

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

```
0001 function id = mysql_getMaxId(conn)
0002 % MYSQL_GETMAXID get the maximum Id from the objs table.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MYSQL_GETMAXID get the maximum Id from the objs table.
0007 %
0008 % CALL: id = mysql_getMaxId()
0009 %
```

```
0010 % VERSION:      $Id: mysql_getMaxId.m,v 1.3 2007/08/31 17:29:22 hewitson Exp $
0011 %
0012 % HISTORY:       24-05-2007 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 q      = 'select max(id) from objs';
0018 curs = exec(conn, q);
0019 curs = fetch(curs);
0020 id    = curs.Data{1};
0021 close(curs);
0022
0023 if isnan(id)
0024     id = 0;
0025 end
0026
0027 % END
```

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

[xydata/xmlparse](matlab:help xydata/xmlparse) - creates a
xydata from a xydata xml file.

[xydata/xydata](matlab:help xydata/xydata) - X-Y data
object class constructor.

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

Generated on Tue 22-Jan-2008 10:39:13 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 1.3 2008/01/07 10:21:53 ingo Exp \$

HISTORY: 30-01-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a provenance object into a string.

This function is called by:

- [char](#) CHAR convert a provenance object into a string.
- [display](#) DISPLAY overload terminal display for provenance objects.

SOURCE CODE

```

0001 function val = get(p, propName)
0002 % GET get provenance properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get provenance properties.
0007 %
0008 % CALL:      creator      = get(p, 'creator');
0009 %            created      = get(p, 'created');
0010 %            ip           = get(p, 'ip');
0011 %            hostname     = get(p, 'hostname');
0012 %            os           = get(p, 'os');
0013 %            matlab_ver   = get(p, 'matlab_version');
0014 %            sigproc_ver  = get(p, 'sigproc_version');
0015 %            ltpda_ver    = get(p, 'ltpda_version');
0016 %
0017 % VERSION:    $Id: get.m,v 1.3 2008/01/07 10:21:53 ingo Exp $
0018 %
0019 % HISTORY:    30-01-2007 M Hewitson
0020 %              Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION      = '$Id: get.m,v 1.3 2008/01/07 10:21:53 ingo Exp $';
0025
0026 % Check if this is a call for parameters
0027 if nargin == 2
0028     if isa(p, 'provenance') && ischar(propName)
0029         in = char(propName);
0030         if strcmp(in, 'Params')
0031             val = plist;
0032             return
0033         elseif strcmp(in, 'Version')
0034             val = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 switch propName
0041     case 'creator'
0042         val = p.creator;
0043     case 'created'
0044         val = p.created;
0045     case 'ip'
0046         val = p.ip;
0047     case 'hostname'
0048         val = p.hostname;
0049     case 'os'

```

```
0050     val = p.os;
0051     case 'matlab_version'
0052         val = p.matlab_version;
0053     case 'sigproc_version'
0054         val = p.sigproc_version;
0055     case 'ltpda_version'
0056         val = p.ltpda_version;
0057     otherwise
0058         error([propName, ' is not a valid provenance property.']);
0059 end
```

[Home](#) > [m](#) > [mysql](#) > mysql_getObjIds.m

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 1.2 2007/11/13 09:44:40 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 [^](#)

```
0001 function ids = mysql_getObjIds(conn, cid)
0002
0003 % MYSQL_GETOBJIDS gets a list of object ids from a given collection id.
0004 %
0005 % Usage: ids = mysql_getObjIds(conn,cid);
0006 %
```



```
0007 % Inputs:
0008 %     conn - a database connection object
0009 %     cid  - a collection id
0010 %
0011 % Outputs:
0012 %     ids  - a list of object ids
0013 %
0014 % M Hewitson 30-08-07
0015 %
0016 % $Id: mysql_getObjIds.m,v 1.2 2007/11/13 09:44:40 hewitson Exp $
0017 %
0018
0019
0020 curs    = exec(conn, sprintf('select obj_ids from collections where id="%d"',
curs));
0021 curs    = fetch(curs);
0022 ids     = str2num(curs.Data{1});
0023 close(curs);
0024
0025
```

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 1.1 2008/01/07 13:46:02 ingo 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for pole properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for pole properties.
0007 %
```

```

0008 % EXAMPLES:      All possible assignment are possible.
0009 %
0010 % VERSION:        $Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $
0011 %
0012 % HISTORY:        31-01-07 M Hewitson
0013 %                  Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'pole') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             A = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             A = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 A = builtin ('subsasgn', A, S, B);
0033

```

[Home](#) > [classes](#) > [@xydata](#) > [private](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@pzmodel](#) > [get.m](#)

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 1.3 2007/10/15 17:16:56 ingo 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:

- [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.

- **xmladd** XMLADD add a pzmodel object to the given node of the input XML DOM

SOURCE CODE

```

0001 function val = get(pzm, propName)
0002 % GET get a pole/zero model property.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get a pole/zero model property.
0007 %
0008 % CALL:          name      = get(pzm, 'name');
0009 %               gain      = get(pzm, 'gain');
0010 %               poles     = get(pzm, 'poles');
0011 %               zeros     = get(pzm, 'zeros');
0012 %               created   = get(pzm, 'created');
0013 %               version   = get(pzm, 'version');
0014 %
0015 % VERSION:       $Id: get.m,v 1.3 2007/10/15 17:16:56 ingo Exp $
0016 %
0017 % HISTORY:       03-04-2007 M Hewitson
0018 %                 Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: get.m,v 1.3 2007/10/15 17:16:56 ingo Exp $';
0023
0024 % Check if this is a call for parameters or for the cvs-version number
0025 if nargin == 2
0026     if isa(pzm, 'pzmodel') && ischar(propName)
0027         in = char(propName);
0028         if strcmp(in, 'Params')
0029             val = plist();
0030             return
0031         elseif strcmp(in, 'Version')
0032             val = VERSION;
0033             return
0034         end
0035     end
0036 end
0037
0038 fields = fieldnames(pzm);
0039
0040 if ~ismember(propName, fields)
0041     error(['### '' propName, '' is not a valid pole/zero model property.']);
0042 else
0043     val = pzm.(propName);
0044 end

```

[Home](#) > [m](#) > [mysql](#) > mysql_getObjType.m

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 1.2 2007/10/17 15:01:45 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

```
0001 function type = mysql_getObjType(conn, id)
0002
0003 % MYSQL_GETOBJTYPE gets the object type associated with the given object id.
0004 %
0005 % Usage: ids = mysql_getObjType(conn,id);
0006 %
```

```
0007 % Inputs:
0008 %     conn - a database connection object
0009 %     id   - an object id
0010 %
0011 % Outputs:
0012 %     type - the type of this object
0013 %
0014 % M Hewitson 30-08-07
0015 %
0016 % $Id: mysql_getObjType.m,v 1.2 2007/10/17 15:01:45 hewitson Exp $
0017 %
0018
0019
0020 curs    = exec(conn, sprintf('select obj_type from objmeta where obj_id="%d"',
id));
0021 curs    = fetch(curs);
0022 type    = curs.Data{1};
0023 close(curs);
0024
0025
```


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 provenance properties.

EXAMPLES:    All possible assignment are possible.

VERSION:     $Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for provenance properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for provenance properties.
0007 %
```

```

0008 % EXAMPLES:      All possible assignment are possible.
0009 %
0010 % VERSION:        $Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'provenance') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             A = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             A = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 A = builtin ('subsasgn', A, S, B);
0033

```

[Home](#) > [classes](#) > [@zero](#) > Contents.m

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 22-Jan-2008 10:39:13 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 1.3 2008/01/11 14:04:43 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

```

0001 function val = get(w, propName)
0002 % GET get specwin object properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get specwin object properties.
0007 %
0008 % CALL:          name      = get(sw, 'name');
0009 %               alpha     = get(sw, 'alpha');
0010 %               psll      = get(sw, 'psll');
0011 %               rov       = get(sw, 'rov');
0012 %               nenbw     = get(sw, 'nenbw');
0013 %               w3db      = get(sw, 'w3db');
0014 %               flatness  = get(sw, 'flatness');
0015 %               ws        = get(sw, 'ws');
0016 %               ws2       = get(sw, 'ws2');
0017 %               win       = get(sw, 'win');
0018 %               created   = get(sw, 'created');
0019 %               created   = get(sw, 'version');
0020 %
0021 % VERSION:       $Id: get.m,v 1.3 2008/01/11 14:04:43 ingo Exp $
0022 %
0023 % HISTORY:       30-01-2007 M Hewitson
0024 %               Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0027
0028 VERSION = '$Id: get.m,v 1.3 2008/01/11 14:04:43 ingo Exp $';
0029
0030 % 'Params' Call
0031 if nargin == 2
0032     if isa(w, 'specwin') && strcmp(propName, 'Params')
0033         val = plist();
0034         return
0035     elseif isa(w, 'specwin') && strcmp(propName, 'Version')
0036         val = VERSION;
0037         return
0038     end
0039 end
0040
0041 switch propName
0042     case 'name'
0043         val = w.name;
0044     case 'alpha'
0045         val = w.alpha;
0046     case 'psll'
0047         val = w.psll;
0048     case 'rov'
0049         val = w.rov;

```

```
0050     case 'nenbw'
0051         val = w.nenbw;
0052     case 'w3db'
0053         val = w.w3db;
0054     case 'flatness'
0055         val = w.flatness;
0056     case 'ws'
0057         val = w.ws;
0058     case 'ws2'
0059         val = w.ws2;
0060     case 'win'
0061         val = w.win;
0062     case 'created'
0063         val = w.created;
0064     case 'version'
0065         val = w.version;
0066     otherwise
0067         error([propName, ' is not a valid specwin property.']);
0068 end
```

[Home](#) > [m](#) > [mysql](#) > mysql_getUserID.m

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 given user name.

CALL: userid = mysql_getUserID(conn, username)

VERSION: \$Id: mysql_getUserID.m,v 1.5 2007/10/17 15:05:23 hewitson Exp \$

HISTORY: 24-05-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [ltpda_obj_submit](#) LTPDA_OBJ_SUBMIT submits the given collection of objects to an LTPDA Repository.

SOURCE CODE

```
0001 function [userid,dbuser] = mysql_getUserID(conn, username)
0002 % MYSQL_GETUSERID gets the user ID number corresponding to the given user name.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MYSQL_GETUSERID gets the user ID number corresponding to the
```



```
0007 %          given user name.
0008 %
0009 % CALL:      userid = mysql_getUserID(conn, username)
0010 %
0011 % VERSION:    $Id: mysql_getUserID.m,v 1.5 2007/10/17 15:05:23 hewitson Exp $
0012 %
0013 % HISTORY:     24-05-2007 M Hewitson
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 q      = sprintf('select id from users where username="%s"', username);
0019 curs = exec(conn, q);
0020 curs = fetch(curs);
0021 userid = curs.Data{1};
0022 close(curs);
0023
0024 % END
```

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 1.1 2008/01/07 13:46:02 ingo 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for pzmodel properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for pzmodel properties.
0007 %
```

```

0008 % EXAMPLES:      All possible assignment are possible.
0009 %
0010 % VERSION:        $Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'pzmodel') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             A = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             A = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 A = builtin ('subsasgn', A, S, B);
0033

```

[Home](#) > [classes](#) > [@zero](#) > [private](#) > Contents.m

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
complex 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 22-Jan-2008 10:39:13 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 1.9 2008/01/14 20:03:52 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:

- **xmladd** XMLADD add a time object to the given node of the input XML DOM

SOURCE CODE

```

0001 function val = get(tt, prop_name)
0002 % GET get time properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get time properties.
0007 %
0008 % CALL:          value = get(tt, 'property');
0009 %
0010 % EXAMPLES:      name           = get(tt, 'name');
0011 %                utc_epoch_milli = get(tt, 'utc_epoch_milli');
0012 %                timezone        = get(tt, 'timezone');
0013 %                timezone_IDs    = get(tt, 'timezone_IDs');
0014 %                timezone_IDs    = get(tt, 'IDs');
0015 %                timeformat      = get(tt, 'timeformat');
0016 %                time_str        = get(tt, 'time_str');
0017 %                created         = get(tt, 'created');
0018 %                version         = get(tt, 'version');
0019 %
0020 % VERSION:       $Id: get.m,v 1.9 2008/01/14 20:03:52 ingo Exp $
0021 %
0022 % HISTORY:       23-07-2007 Diepholz
0023 %                Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026
0027 VERSION = '$Id: get.m,v 1.9 2008/01/14 20:03:52 ingo Exp $';
0028
0029 % Check if this is a call for parameters or for the cvs-version number
0030 if nargin == 2
0031     if isa(tt, 'time') && ischar(prop_name)
0032         in = char(prop_name);
0033         if strcmp(in, 'Params')
0034             val = plist();
0035             return
0036         elseif strcmp(in, 'Version')
0037             val = VERSION;
0038             return
0039         end
0040     end
0041 end
0042
0043 fields = fieldnames(tt);
0044
0045 fields{end+1} = 'timezone_IDs';
0046
0047 if strcmpi(prop_name, 'timezone_IDs') || strcmpi(prop_name, 'IDs')

```

```
0048     val = char(java.util.TimeZone.getAvailableIDs);
0049     return
0050 end
0051
0052 if ~ismember(prop_name, fields)
0053     error(['### '' prop_name, '' is not a valid time property.']);
0054 else
0055     val = tt.(prop_name);
0056 end
0057
0058
```

[Home](#) > [m](#) > [mysql](#) > mysql_getXdoc.m

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 1.2 2007/10/17 15:04:27 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function xdoc = mysql_getXdoc(conn, id)
0002
0003 % MYSQL_GETXDOC retrieves an object with given id from the LTPDA
```



```

0004 % repository specified by the input database connection. The object is
0005 % converted from its XML text format to an Xdoc. This can then be converted
0006 % into an object using the appropriate object constructor.
0007 %
0008 % Usage: xdoc = mysql_getXdoc(conn, id)
0009 %
0010 % Inputs:
0011 %         conn - a database connection object
0012 %         id   - the object id
0013 %
0014 % Outputs:
0015 %         xdoc - an Xdoc representation of the object.
0016 %
0017 % M Hewitson 30-08-07
0018 %
0019 % $Id: mysql_getXdoc.m,v 1.2 2007/10/17 15:04:27 hewitson Exp $
0020 %
0021 %
0022
0023
0024 curs    = exec(conn, sprintf('select xml from objs where id="%d"', id));
0025 curs    = fetch(curs);
0026 objTxt  = char([curs.Data{1}].');
0027 close(curs);
0028
0029
0030 % convert to Java string
0031 str = java.lang.String(objTxt);
0032 % open stream on this string
0033 stream = java.io.StringBufferInputStream(str);
0034 % make parser
0035 factory = javaMethod('newInstance', ...
0036     'javax.xml.parsers.DocumentBuilderFactory');
0037 builder = factory.newDocumentBuilder;
0038
0039 xdoc = builder.parse(stream);
0040
0041 % END

```

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 1.2 2008/01/11 14:04:43 ingo 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for specwin properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for specwin properties.
0007 %
```

```

0008 % EXAMPLES:      All possible assignment are possible.
0009 %
0010 % VERSION:        $Id: subsasgn.m,v 1.2 2008/01/11 14:04:43 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsasgn.m,v 1.2 2008/01/11 14:04:43 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'specwin') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             A = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             A = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 A = builtin ('subsasgn', A, S, B);
0033

```

[Home](#) > [m](#) > [analyses](#) > [stuttgart](#) > Contents.m

Contents

PURPOSE

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/analyses/stuttgart %%%%%%%%%%
```

SYNOPSIS

This is a script file.

DESCRIPTION

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/analyses/stuttgart %%%%%%%%%%
```

```
    <a href="matlab:help m/analyses/stuttgart/ltpda_diff_acc">m/analyses/stuttgart/  
ltpda_diff_acc</a>    - (No help available)  
    <a href="matlab:help m/analyses/stuttgart/ltpda_dkalman">m/analyses/stuttgart/  
ltpda_dkalman</a>    - (No help available)
```

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@tsdata](#) > [get.m](#)

get

PURPOSE

GET get tsdata properties.

SYNOPSIS

function val = get(ts, propName)

DESCRIPTION

GET get tsdata properties.

%%%

DESCRIPTION: GET get tsdata properties.

```
CALL:      name = get(ts, 'name');
           fs = get(ts, 'fs');
           x = get(ts, 'x');
           y = get(ts, 'y');
           nsecs = get(ts, 'nsecs');
           t0 = get(ts, 't0');
           xunits = get(ts, 'xunits');
           yunits = get(ts, 'yunits');
           created = get(ts, 'created');
           version = get(ts, 'version');
```

VERSION: \$Id: get.m,v 1.7 2007/11/13 16:03:20 ingo Exp \$

HISTORY: 30-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a param object into a string.

This function is called by:

- [char](#) CHAR convert a param object into a string.

- **display** DISPLAY overloads display functionality for tsdata objects.
- **xmladd** XMLADD add a tsdata object to the given node of the input XML DOM

SOURCE CODE

```

0001 function val = get(ts, propName)
0002 % GET get tsdata properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get tsdata properties.
0007 %
0008 % CALL:      name = get(ts, 'name');
0009 %           fs = get(ts, 'fs');
0010 %           x = get(ts, 'x');
0011 %           y = get(ts, 'y');
0012 %           nsecs = get(ts, 'nsecs');
0013 %           t0 = get(ts, 't0');
0014 %           xunits = get(ts, 'xunits');
0015 %           yunits = get(ts, 'yunits');
0016 %           created = get(ts, 'created');
0017 %           version = get(ts, 'version');
0018 %
0019 % VERSION: $Id: get.m,v 1.7 2007/11/13 16:03:20 ingo Exp $
0020 %
0021 % HISTORY: 30-01-07 M Hewitson
0022 %           Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025 %
0026 VERSION = '$Id: get.m,v 1.7 2007/11/13 16:03:20 ingo Exp $';
0027
0028 % Check if this is a call for parameters
0029 if nargin == 2
0030     if isa(ts, 'tsdata') && ischar(propName)
0031         in = char(propName);
0032         if strcmp(in, 'Params')
0033             val = plist;
0034             return
0035         elseif strcmp(in, 'Version')
0036             val = VERSION;
0037             return
0038         end
0039     end
0040 end
0041
0042 switch propName
0043     case 'name'
0044         val = ts.name;
0045     case 'created'
0046         val = ts.created;

```

```
0047     case 'version'
0048         val = ts.version;
0049     case 'fs'
0050         val = ts.fs;
0051     case 'x'
0052         val = ts.x;
0053     case 'y'
0054         val = ts.y;
0055     case 'xunits'
0056         val = ts.xunits;
0057     case 'yunits'
0058         val = ts.yunits;
0059     case 'nsecs'
0060         val = ts.nsecs;
0061     case 't0'
0062         val = ts.t0;
0063     otherwise
0064         error([propName, ' is not a valid tsdata property.']);
0065 end
```

[Home](#) > [classes](#) > [@time](#) > subsasgn.m

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 1.1 2008/01/07 13:46:02 ingo 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for time properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for time properties.
0007 %
```



```

0008 % EXAMPLES:      All possible assignment are possible.
0009 %
0010 % VERSION:        $Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'time') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             A = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             A = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 A = builtin ('subsasgn', A, S, B);
0033

```

[Home](#) > [m](#) > [etc](#) > Contents.m

Contents

PURPOSE

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/etc %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

SYNOPSIS

This is a script file.

DESCRIPTION

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/etc %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    <a href="matlab:help m/etc/ao_fcn_template">m/etc/ao_fcn_template</a> - F00  
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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@xydata](#) > get.m

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 1.4 2007/11/13 16:03:20 ingo Exp \$

HISTORY: 30-01-2007 Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a param object into a string.

This function is called by:

- [char](#) CHAR convert a param object into a string.
- [display](#) DISPLAY overloads display functionality for xydata objects.
- [xmladd](#) XMLADD add a xydata object to the given node of the input XML DOM

SOURCE CODE

```

0001 function val = get(xy, propName)
0002 % GET gets xydata properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET gets xydata properties.
0007 %
0008 % CALL:      name      = get(xy, 'name');
0009 %           x          = get(xy, 'x');
0010 %           y          = get(xy, 'y');
0011 %           xunits     = get(xy, 'xunits');
0012 %           yunits     = get(xy, 'yunits');
0013 %           created     = get(xy, 'created');
0014 %           version     = get(xy, 'version');
0015 %
0016 % VERSION: $Id: get.m,v 1.4 2007/11/13 16:03:20 ingo Exp $
0017 %
0018 % HISTORY: 30-01-2007 Hewitson
0019 %           Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 VERSION = '$Id: get.m,v 1.4 2007/11/13 16:03:20 ingo Exp $';
0024
0025 % Check if this is a call for parameters
0026 if nargin == 2
0027     if isa(xy, 'xydata') && ischar(propName)
0028         in = char(propName);
0029         if strcmp(in, 'Params')
0030             val = plist;
0031             return
0032         elseif strcmp(in, 'Version')
0033             val = VERSION;
0034             return
0035         end
0036     end
0037 end
0038
0039 switch propName
0040     case 'name'
0041         val = xy.name;
0042     case 'created'
0043         val = xy.created;
0044     case 'version'
0045         val = xy.version;
0046     case 'x'
0047         val = xy.x;
0048     case 'y'
0049         val = xy.y;

```

```
0050     case 'xunits'  
0051         val = xy.xunits;  
0052     case 'yunits'  
0053         val = xy.yunits;  
0054     otherwise  
0055         error([propName, ' is not a valid xydata property.']);  
0056 end
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@cdata](#) > ne.m

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')

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 1.2 2007/11/28 13:03:32 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

```

0001 function result = ne(c1, c2, varargin)
0002 % NE overloads the ~= operator for cdata objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for cdata objects.
0007 %
0008 % CALL:          result = ne(c1,c2)
0009 %               result = ne(c1,c2, 'property1', 'property2')
0010 %               result = ne(c1,c2, 'class/property', 'class/property')
0011 %
0012 % INPUTS:        c1,c2 - input cdata objects
0013 %
0014 % OUTPUTS:       If the two cdata objects are considered equal, result == 0,
0015 %               otherwise, result == 1.
0016 %
0017 % VERSION:       $Id: ne.m,v 1.2 2007/11/28 13:03:32 ingo Exp $
0018 %
0019 % HISTORY:       29-08-2007 M Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: ne.m,v 1.2 2007/11/28 13:03:32 ingo Exp $';
0025
0026 % Check if this is a call for parameters
0027 if nargin == 2
0028     if isa(c1, 'cdata') && ischar(c2)
0029         in = char(c2);
0030         if strcmp(in, 'Params')
0031             result = plist;
0032             return
0033         elseif strcmp(in, 'Version')
0034             result = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 if eq(c1,c2, varargin{:})
0041     result = 0;
0042 else
0043     result = 1;
0044 end
0045
0046 % END

```

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 1.1 2008/01/07 13:46:02 ingo 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for timeformat properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for timeformat properties.
0007 %
```



```

0008 % EXAMPLES:      All possible assignment are possible.
0009 %
0010 % VERSION:        $Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'timeformat') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             A = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             A = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 A = builtin ('subsasgn', A, S, B);
0033

```

[Home](#) > [m](#) > [gui](#) > [ao_browser](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@xyzdata](#) > get.m

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 1.1 2007/12/24 19:47:06 hewitson 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

```

0001 function val = get(xyz, propName)
0002 % GET gets xyzdata properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET gets xyzdata properties.
0007 %
0008 % CALL:      name      = get(xyz, 'name');
0009 %           x          = get(xyz, 'x');
0010 %           y          = get(xyz, 'y');
0011 %           z          = get(xyz, 'z');
0012 %           xunits     = get(xyz, 'xunits');
0013 %           yunits     = get(xyz, 'yunits');
0014 %           zunits     = get(xyz, 'zunits');
0015 %           created     = get(xyz, 'created');
0016 %           version     = get(xyz, 'version');
0017 %
0018 % VERSION: $Id: get.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $
0019 %
0020 % HISTORY: 30-01-2007 Hewitson
0021 %           Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: get.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $';
0026
0027 % Check if this is a call for parameters
0028 if nargin == 2
0029     if ischar(propName)
0030         in = char(propName);
0031         if strcmp(in, 'Params')
0032             val = plist;
0033             return
0034         elseif strcmp(in, 'Version')
0035             val = VERSION;
0036             return
0037         end
0038     end
0039 end
0040
0041 switch propName
0042     case 'name'
0043         val = xyz.name;
0044     case 'created'
0045         val = xyz.created;
0046     case 'version'
0047         val = xyz.version;
0048     case 'x'
0049         val = xyz.x;
0050     case 'y'
0051         val = xyz.y;
0052     case 'z'

```

```
0053     val = xyz.z;  
0054     case 'xunits'  
0055         val = xyz.xunits;  
0056     case 'yunits'  
0057         val = xyz.yunits;  
0058     case 'zunits'  
0059         val = xyz.zunits;  
0060     otherwise  
0061         error([propName, ' is not a valid xyzdata property.']);  
0062 end  
0063  
0064
```

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')

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 1.2 2007/11/28 13:03:32 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

```

0001 function result = ne(c1, c2, varargin)
0002 % NE overloads the ~= operator for fsdata objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for fsdata objects.
0007 %
0008 % CALL:          result = ne(c1,c2)
0009 %               result = ne(c1,c2, 'property1', 'property2')
0010 %               result = ne(c1,c2, 'class/property', 'class/property')
0011 %
0012 % INPUTS:        c1,c2 - input fsdata objects
0013 %
0014 % OUTPUTS:       If the two fsdata objects are considered equal, result == 0,
0015 %               otherwise, result == 1.
0016 %
0017 % VERSION:       $Id: ne.m,v 1.2 2007/11/28 13:03:32 ingo Exp $
0018 %
0019 % HISTORY:       29-08-2007 M Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: ne.m,v 1.2 2007/11/28 13:03:32 ingo Exp $';
0025
0026 % Check if this is a call for parameters
0027 if nargin == 2
0028     if isa(c1, 'fsdata') && ischar(c2)
0029         in = char(c2);
0030         if strcmp(in, 'Params')
0031             result = plist;
0032             return
0033         elseif strcmp(in, 'Version')
0034             result = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 if eq(c1,c2, varargin{:})
0041     result = 0;
0042 else
0043     result = 1;
0044 end
0045
0046 % END

```

[Home](#) > [classes](#) > [@timespan](#) > subsasgn.m

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 1.1 2008/01/07 13:46:02 ingo 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for timespan properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for timespan properties.
0007 %
```



```

0008 % EXAMPLES:      All possible assignment are possible.
0009 %
0010 % VERSION:        $Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'timespan') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             A = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             A = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 A = builtin ('subsasgn', A, S, B);
0033

```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > Contents.m

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 graphicuitable component in a figure window.
    <a href="matlab:help m/gui/ltpdaRepoGUI/repogui">m/gui/ltpdaRepoGUI/repogui</a>
        - a GUI to interact with an LTPDA Repository.
    <a href="matlab:help m/gui/ltpdaRepoGUI/sqlResultsGUI">m/gui/ltpdaRepoGUI/sqlResultsGUI</a>
        - shows the results of a query in a new window.
    <a href="matlab:help m/gui/ltpdaRepoGUI/uitabdemo">m/gui/ltpdaRepoGUI/uitabdemo</a>
        - This script demonstrates 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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@zero](#) > get.m

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 1.4 2007/10/16 17:37:21 ingo Exp \$

HISTORY: 30-01-2007 Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a zero object into a string.

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.

- **xmladd** XMLADD add a zero object to the given node of the input XML DOM

SOURCE CODE

```

0001 function val = get(p, propName)
0002 % GET get zero properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get zero properties.
0007 %
0008 % CALL:          name      = get(z, 'name');
0009 %              f          = get(z, 'f');
0010 %              q          = get(z, 'q');
0011 %              ri         = get(z, 'ri');
0012 %              version    = get(z, 'version');
0013 %              created    = get(z, 'created');
0014 %
0015 % VERSION:       $Id: get.m,v 1.4 2007/10/16 17:37:21 ingo Exp $
0016 %
0017 % HISTORY:       30-01-2007 Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: get.m,v 1.4 2007/10/16 17:37:21 ingo Exp $';
0023
0024 % Check if this is a call for parameters or for the cvs-version number
0025 if nargin == 2
0026     if isa(p, 'zero') && ischar(propName)
0027         in = char(propName);
0028         if strcmp(in, 'Params')
0029             val = plist();
0030             return
0031         elseif strcmp(in, 'Version')
0032             val = VERSION;
0033             return
0034         end
0035     end
0036 end
0037
0038 switch propName
0039     case 'name'
0040         val = p.name;
0041     case 'f'
0042         val = p.f;
0043     case 'q'
0044         val = p.q;
0045     case 'ri'
0046         val = p.ri;
0047     case 'version'

```

```
0048     val = p.version;  
0049     case 'created'  
0050         val = p.created;  
0051     otherwise  
0052         error([propName, ' is not a valid zero property.']);  
0053 end
```

[Home](#) > [classes](#) > [@history](#) > ne.m

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')

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 1.2 2007/12/05 16:26:22 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

```

0001 function result = ne(w1,w2, varargin)
0002
0003 % NE overloads the ~= operator for history objects.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: NE overloads the ~= operator for history objects.
0008 %
0009 % CALL:      result = ne(h1,h2)
0010 %           result = ne(h1,h2, 'property1', 'property2')
0011 %           result = ne(h1,h2, 'class/property', 'class/property')
0012 %
0013 % INPUTS:    h1,h2 - input history objects
0014 %
0015 % OUTPUTS:   If the two history objects are considered equal, result == 0,
0016 %           otherwise, result == 1.
0017 %
0018 % VERSION:   $Id: ne.m,v 1.2 2007/12/05 16:26:22 ingo Exp $
0019 %
0020 % HISTORY:   29-08-2007 M Hewitson
0021 %           Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: ne.m,v 1.2 2007/12/05 16:26:22 ingo Exp $';
0026
0027 % Check if this is a call for parameters
0028 if nargin == 2
0029     if isa(w1, 'history') && ischar(w2)
0030         in = char(w2);
0031         if strcmp(in, 'Params')
0032             result = plist;
0033             return
0034         elseif strcmp(in, 'Version')
0035             result = VERSION;
0036             return
0037         end
0038     end
0039 end
0040
0041 if eq(w1, w2, varargin{:})
0042     result = 0;
0043 else
0044     result = 1;
0045 end
0046
0047 % END

```

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 1.3 2007/11/23 12:11:08 ingo 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for for tsdata properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for tsdata properties.
0007 %
```



```

0008 % EXAMPLES:      All possible assignments are possible.
0009 %
0010 % VERSION:        $Id: subsasgn.m,v 1.3 2007/11/23 12:11:08 ingo Exp $
0011 %
0012 % HISTORY:        31-01-07 M Hewitson
0013 %                  Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsasgn.m,v 1.3 2007/11/23 12:11:08 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'tsdata') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             A = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             A = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032
0033 A = builtin ('subsasgn', A, S, B);
0034
0035
0036 % END

```

Contents

PURPOSE

%% path: m/gui/ltpdaRepoGUI/callbacks %%%

SYNOPSIS

This is a script file.

DESCRIPTION

%% path: m/gui/ltpdaRepoGUI/callbacks %%%

[m/gui/ltpdaRepoGUI/callbacks/buildConditions](matlab:help m/gui/ltpdaRepoGUI/callbacks/buildConditions) - before deleting, record the table selection

[m/gui/ltpdaRepoGUI/callbacks/buildquery](matlab:help m/gui/ltpdaRepoGUI/callbacks/buildquery) - Get number of conditions

m/gui/ltpdaRepoGUI/callbacks/cb_condPlusBtn - (No help available)

m/gui/ltpdaRepoGUI/callbacks/cb_condSubtractBtn - (No help available)

[m/gui/ltpdaRepoGUI/callbacks/cb_connectBtn](matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_connectBtn) - Get conn

[m/gui/ltpdaRepoGUI/callbacks/cb_dbSelect](matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_dbSelect) - Get selected DB

[m/gui/ltpdaRepoGUI/callbacks/cb_executeQuery](matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_executeQuery) - Get connection

[m/gui/ltpdaRepoGUI/callbacks/cb_fieldSelect](matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_fieldSelect) - Build query

[m/gui/ltpdaRepoGUI/callbacks/cb_getDBs](matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_getDBs) - conn

m/gui/ltpdaRepoGUI/callbacks/cb_mainfigClose - (No help available)

[m/gui/ltpdaRepoGUI/callbacks/cb_refreshObjsList](matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_refreshObjsList) - get a list of objects

[m/gui/ltpdaRepoGUI/callbacks/cb_retrieveBtn](matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_retrieveBtn) - Get connection

[m/gui/ltpdaRepoGUI/callbacks/cb_submitBtn](matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_submitBtn) - get object list

[m/gui/ltpdaRepoGUI/callbacks/cb_tableSelect](matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_tableSelect) - Get conn

[m/gui/ltpdaRepoGUI/callbacks/cb_workspaceObjsList](matlab:help m/gui/ltpdaRepoGUI/callbacks/cb_workspaceObjsList) - get the selected line

[m/gui/ltpdaRepoGUI/callbacks/getFields](matlab:help m/gui/ltpdaRepoGUI/callbacks/getFields) - the selected table

[m/gui/ltpdaRepoGUI/callbacks/getTables](matlab:help m/gui/ltpdaRepoGUI/callbacks/getTables) - 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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > getFields.m

getFields

PURPOSE

get the selected table

SYNOPSIS

```
function fields = getFields(conn)
```

DESCRIPTION

get the selected table

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [buildConditions](#)
- [cb_tableSelect](#)

SOURCE CODE

```
0001 function fields = getFields(conn)
0002
0003
0004 % get the selected table
0005 tblSel = findobj('Tag', 'tableSelect');
0006 tbls   = get(tblSel, 'String');
0007 tbl    = tbls{get(tblSel, 'Value')};
0008
0009 if ~isempty(conn)
0010     % get the databases
```

```
0011    q = sprintf('describe %s', tbl);
0012    curs = exec(conn, q);
0013    curs = fetch(curs);
0014
0015    fields = curs.Data(:,1);
0016    close(curs);
0017 else
0018     fields = {'-'};
0019 end
0020
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@mfir](#) > ne.m

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,f1)
 result = ne(f1,f2, 'property1', 'property2')
 result = ne(f1,f2, 'class/property', 'class/property')

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 1.2 2007/12/06 17:09:15 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

```

0001 function result = ne(f1, f2, varargin)
0002
0003 % NE overloads the ~= operator for mfir objects.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: NE overloads the ~= operator for mfir objects.
0008 %
0009 % CALL:      result = ne(f1,f1)
0010 %            result = ne(f1,f2, 'property1', 'property2')
0011 %            result = ne(f1,f2, 'class/property', 'class/property')
0012 %
0013 % INPUTS:    f1,f2 - input mfir objects
0014 %
0015 % OUTPUTS:   If the two mfir objects are considered equal, result == 0,
0016 %            otherwise, result == 1.
0017 %
0018 % VERSION:   $Id: ne.m,v 1.2 2007/12/06 17:09:15 ingo Exp $
0019 %
0020 % HISTORY:   29-08-2007 M Hewitson
0021 %            Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: ne.m,v 1.2 2007/12/06 17:09:15 ingo Exp $';
0026
0027 % Check if this is a call for parameters
0028 if nargin == 2
0029     if isa(f1, 'mfir') && ischar(f2)
0030         in = char(f2);
0031         if strcmp(in, 'Params')
0032             result = plist;
0033             return
0034         elseif strcmp(in, 'Version')
0035             result = VERSION;
0036             return
0037         end
0038     end
0039 end
0040
0041 if eq(f1, f2, varargin{:})
0042     result = 0;
0043 else
0044     result = 1;
0045 end
0046
0047 % END

```

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 1.3 2007/11/23 12:11:09 ingo 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for xydata properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for xydata properties.
0007 %
```



```

0008 % EXAMPLES:      All possible assignment are possible.
0009 %
0010 % VERSION:       $Id: subsasgn.m,v 1.3 2007/11/23 12:11:09 ingo Exp $
0011 %
0012 % HISTORY:       31-01-07 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsasgn.m,v 1.3 2007/11/23 12:11:09 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'xydata') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             A = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             A = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032
0033 A = builtin ('subsasgn', A, S, B);
0034

```

[Home](#) > [m](#) > [gui](#) > [pzmodel_designer](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@history](#) > [getNodes.m](#)

getNodes

PURPOSE

GETNODES converts a history object to a nodes structure suitable for plotting as a tree.

SYNOPSIS

function [n,a,nodes] = getNodes(h, n, pn, a, nodes)

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 1.10 2007/12/04 18:16:11 ingo 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.
- [set](#) SET set a history property.

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

```

0001 function [n,a,nodes] = getNodes(h, n, pn, a, nodes)
0002 % GETNODES converts a history object to a nodes structure suitable for plotting
as a tree.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GETNODES converts a history object to a nodes structure suitable
0007 %               for plotting as a tree.
0008 %
0009 % CALL:        [n,a, nodes] = getNodes(hist, [], 0, 1, []);
0010 %
0011 % OUTPUT:      n:      A vector of parent pointers
0012 %               a:      Number of nodes
0013 %               nodes: Struct of the nodes
0014 %
0015 % VERSION:     $Id: getNodes.m,v 1.10 2007/12/04 18:16:11 ingo Exp $
0016 %
0017 % HISTORY:     02-02-2007 M Hewitson
0018 %               Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: getNodes.m,v 1.10 2007/12/04 18:16:11 ingo Exp $';
0023
0024 % Check if this is a call for parameters
0025 if nargin == 2
0026     if isa(h, 'history') && ischar(n)
0027         in = char(n);
0028         if strcmp(in, 'Params')
0029             n = plist;
0030             return
0031         elseif strcmp(in, 'Version')
0032             n = VERSION;
0033             return
0034         end
0035     end
0036 end
0037
0038
0039 % set my node to next number
0040 nn = get(h, 'n');
0041 if nn<0
0042     h = set(h, 'n', a);
0043 end
0044

```

```

0045 % set my parent node
0046 nn = get(h, 'pn');
0047 if nn<0
0048     h = set(h, 'pn', pn);
0049     n = [n pn];
0050     nodes(a).pn      = pn;
0051     nodes(a).invars = get(h, 'invars');
0052     nodes(a).n       = get(h, 'n');
0053     nodes(a).names   = get(h, 'name');
0054     nodes(a).params  = char(get(h, 'plist'));
0055     nodes(a).pl      = get(h, 'plist');
0056     nodes(a).h       = h;
0057 end
0058 % disp(['node: ' num2str(get(h, 'pn'))]);
0059
0060 % I need my parent node
0061 pn = get(h, 'n');
0062
0063 % get my children nodes
0064 ihs = get(h, 'inhists');
0065
0066 % Now decide what to do with my children
0067 if isa(ihs, 'history')
0068     for i=1:length(ihs)
0069         % get child number
0070         nn = get(ihs(i), 'n');
0071         % if this child is not set
0072         if nn < 0
0073             % set it
0074             a = a + 1;
0075             [n,a, nodes] = getNodes(ihs(i), n, pn, a, nodes);
0076         else % else go back to my parent
0077             [n,a, nodes] = getNodes(h, n, get(h, 'pn'), a, nodes);
0078         end
0079     end
0080 end
0081

```

[Home](#) > [classes](#) > [@miir](#) > ne.m

ne

PURPOSE

NE overloads the ~= operator for miir objects.

SYNOPSIS

function result = ne(f1, f2, varargin)

DESCRIPTION

NE overloads the ~= operator for miir objects.

%%%

DESCRIPTION: NE overloads the ~= operator for miir objects.

CALL: result = ne(f1,f1)

INPUTS: f1,f2 - input miir objects

OUTPUTS: If the two miir objects are considered equal, result == 0,
otherwise, result == 1.

VERSION: \$Id: ne.m,v 1.2 2007/12/06 17:09:15 ingo Exp \$

HISTORY: 29-08-2007 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a miir object into a string.
- [eq](#) EQ overloads the == operator for miir objects.

This function is called by:

SOURCE CODE

```
0001 function result = ne(f1, f2, varargin)
```

```

0002
0003 % NE overloads the ~= operator for miir objects.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: NE overloads the ~= operator for miir objects.
0008 %
0009 % CALL:          result = ne(f1,f1)
0010 %
0011 % INPUTS:        f1,f2 - input miir objects
0012 %
0013 % OUTPUTS:       If the two miir objects are considered equal, result == 0,
0014 %                otherwise, result == 1.
0015 %
0016 % VERSION:       $Id: ne.m,v 1.2 2007/12/06 17:09:15 ingo Exp $
0017 %
0018 % HISTORY:       29-08-2007 M Hewitson
0019 %                Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 VERSION = '$Id: ne.m,v 1.2 2007/12/06 17:09:15 ingo Exp $';
0024
0025 % Check if this is a call for parameters
0026 if nargin == 2
0027     if isa(f1, 'miir') && ischar(f2)
0028         in = char(f2);
0029         if strcmp(in, 'Params')
0030             result = plist;
0031             return
0032         elseif strcmp(in, 'Version')
0033             result = VERSION;
0034             return
0035         end
0036     end
0037 end
0038
0039 if eq(f1, f2, varargin{:})
0040     result = 0;
0041 else
0042     result = 1;
0043 end
0044
0045 % END

```

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 1.1 2007/12/24 19:47:06 hewitson Exp \$

HISTORY: 24-12-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for xyzdata properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for xyzdata properties.
0007 %
0008 % EXAMPLES: All possible assignment are possible.
0009 %
```



```

0010 % VERSION:      $Id: subsasgn.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $
0011 %
0012 % HISTORY:      24-12-07 M Hewitson
0013 %              Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsasgn.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'xyzdata') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             A = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             A = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032
0033 A = builtin ('subsasgn', A, S, B);
0034

```

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>
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>
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>
a> - Convert complex pole/zero into frequency/Q pole/zero
representation.
  <a href="matlab:help m/helper/ltpda_saveobj">m/helper/ltpda_saveobj</a>
a> - saves an object to a file.
  <a href="matlab:help m/helper/ltpda_str2cells">m/helper/ltpda_str2cells</a>
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>
a> - convert a set of input strings to a cell array.
  <a href="matlab:help m/helper/ltpda_structcat">m/helper/ltpda_structcat</a>
a> - concatonate structures to make one large structure.
  <a href="matlab:help m/helper/ltpda_structidx">m/helper/ltpda_structidx</a>
a> - Index a structure.
  <a href="matlab:help m/helper/ltpda_unwrapdeg">m/helper/ltpda_unwrapdeg</a>
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:

[Home](#) > [classes](#) > [@history](#) > getNodes_plot.m

getNodes_plot

PURPOSE

GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.

SYNOPSIS

function [n,a,nodes] = getNodes_plot(h, n, pn, a, nodes, depth, MAX_DEPTH)

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 1.3 2007/12/04 18:16:11 ingo 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.
- [set](#) SET set a history property.

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

```

0001 function [n,a,nodes] = getNodes_plot(h, n, pn, a, nodes, depth, MAX_DEPTH)
0002 % GETNODES_PLOT converts a history object to a nodes structure suitable for
0003 % plotting as a tree.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GETNODES_PLOT converts a history object to a nodes structure
0007 %               suitable for plotting as a tree.
0008 %
0009 % CALL:        [n,a, nodes] = getNodes(hist, [], 0, 1, [], 0, 2);
0010 %
0011 % OUTPUT:      n:      A vector of parent pointers
0012 %              a:      Number of nodes
0013 %              nodes: Struct of the nodes
0014 %
0015 % REMARK:      This version is equivalent to getNodes with the different that
0016 %              this function only return the nodes up to the MAX_DEPTH.
0017 %
0018 % VERSION:     $Id: getNodes_plot.m,v 1.3 2007/12/04 18:16:11 ingo Exp $
0019 %
0020 % HISTORY:     14-06-07 M Hewitson
0021 %              Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: getNodes_plot.m,v 1.3 2007/12/04 18:16:11 ingo Exp $';
0026
0027 % Check if this is a call for parameters
0028 if nargin == 2
0029     if isa(h, 'history') && ischar(n)
0030         in = char(n);
0031         if strcmp(in, 'Params')
0032             n = plist;
0033             return
0034         elseif strcmp(in, 'Version')
0035             n = VERSION;
0036             return
0037         end
0038     end
0039 end
0040
0041 % set my node to next number

```

```

0042 nn = get(h, 'n');
0043 if nn<0
0044     h = set(h, 'n', a);
0045 end
0046
0047 % set my parent node
0048 nn = get(h, 'pn');
0049 if nn<0
0050     h = set(h, 'pn', pn);
0051     n = [n pn];
0052     nodes(a).pn      = pn;
0053     nodes(a).invars = get(h, 'invars');
0054     nodes(a).n       = get(h, 'n');
0055     nodes(a).names   = get(h, 'name');
0056     nodes(a).params = char(get(h, 'plist'));
0057     nodes(a).pl      = get(h, 'plist');
0058
0059     if depth == MAX_DEPTH
0060         nodes(a).names = '...';
0061         nodes(a).params = '';
0062         nodes(a).invars = {};
0063     end
0064
0065 end
0066
0067 % I need my parent node
0068 pn = get(h, 'n');
0069
0070 % get my children nodes
0071 ihs = get(h, 'inhists');
0072
0073 % Now decide what to do with my children
0074 if isa(ihs, 'history')
0075     for i=1:length(ihs)
0076         % get child number
0077         nn = get(ihs(i), 'n');
0078         % if this child is not set
0079         if nn < 0
0080             if depth < MAX_DEPTH
0081
0082                 % set it
0083                 a = a + 1;
0084                 [n,a, nodes] = getNodes_plot(ihs(i), n, pn, a, nodes, depth+1,
MAX_DEPTH);
0085
0086             end
0087         else % else go back to my parent
0088             [n,a, nodes] = getNodes_plot(h, n, get(h, 'pn'), a, nodes, depth,
MAX_DEPTH);
0089         end
0090     end

```

```
0091  
0092 end  
0093
```

Generated on Tue 22-Jan-2008 10:39:13 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 = eq(p1,p2, 'property1', 'property2')
 result = eq(p1,p2, 'class/property', 'class/property')

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 1.2 2007/12/11 17:27:09 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


```

0001 function result = ne(p1,p2, varargin)
0002 % NE overloads the ~= operator for param objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for param objects.
0007 %
0008 % CALL:      result = ne(p1,p2)
0009 %            result = eq(p1,p2, 'property1', 'property2')
0010 %            result = eq(p1,p2, 'class/property', 'class/property')
0011 %
0012 % INPUTS:    p1,p2 - input param objects
0013 %
0014 % OUTPUTS:   If the two param objects are considered equal, result == 0,
0015 %            otherwise, result == 1.
0016 %
0017 % VERSION:   $Id: ne.m,v 1.2 2007/12/11 17:27:09 ingo Exp $
0018 %
0019 % HISTORY:   29-08-2007 M Hewitson
0020 %            Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: ne.m,v 1.2 2007/12/11 17:27:09 ingo Exp $';
0025
0026 % Check if this is a call for parameters
0027 if nargin == 2
0028     if isa(p1, 'param') && ischar(p2)
0029         in = char(p2);
0030         if strcmp(in, 'Params')
0031             result = plist;
0032             return
0033         elseif strcmp(in, 'Version')
0034             result = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 if eq(p1, p2, varargin{:})
0041     result = 0;
0042 else
0043     result = 1;
0044 end
0045
0046 % END

```

[Home](#) > [classes](#) > [@zero](#) > subsasgn.m

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 1.1 2008/01/07 13:46:02 ingo 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

```
0001 function A = subsasgn(A, S, B)
0002 % SUBSASGN define index assignment for zero properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSASGN define index assignment for zero properties.
0007 %
```

```

0008 % EXAMPLES:      All possible assignment are possible.
0009 %
0010 % VERSION:        $Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsasgn.m,v 1.1 2008/01/07 13:46:02 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'zero') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             A = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             A = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 A = builtin ('subsasgn', A, S, B);
0033

```

[Home](#) > [m](#) > [math](#) > Contents.m

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [getTables.m](#)

getTables

PURPOSE

SYNOPSIS

```
function tables = getTables(conn)
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [buildConditions](#)

SOURCE CODE

```
0001 function tables = getTables(conn)
0002
0003 mainfig = findobj('Tag', 'LTPDARepomainfig');
0004
0005 % Get conn
0006 conn = getappdata(mainfig, 'connection');
0007
0008 if ~isempty(conn)
0009     % get the databases
0010     q = 'show tables';
0011     curs = exec(conn, q);
0012     curs = fetch(curs);
0013
0014     tables = curs.Data;
```

```
0015     close(curs);  
0016 else  
0017     tables = {'-'};  
0018 end  
0019
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [mysql](#) > Contents.m

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 A0 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 A0 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:

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [getWorkspaceObjs.m](#)

getWorkspaceObjs

PURPOSE

get base workspace variables

SYNOPSIS

```
function objs = getWorkspaceObjs()
```

DESCRIPTION

get base workspace variables

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [cb_refreshObjsList](#) get a list of objects
- [cb_refreshObjsListQL](#) get a list of objects

SOURCE CODE

```
0001 function objs = getWorkspaceObjs()
0002
0003 % get base workspace variables
0004 ws_vars = evalin('base','whos');
0005
0006 objs = [];
0007 for j=1:length(ws_vars)
0008
0009     cmd = sprintf('obj = evalin(''base'', ''%s'');', ws_vars(j).name);
0010     eval(cmd)
0011     if ltpda_isuserobject(obj)
0012         objs = [objs ws_vars(j)];
```



```
0013     end
0014
0015 end
0016
0017
0018
0019
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@pole](#) > ne.m

ne

PURPOSE

NE overloads the ~= operator for pole objects.

SYNOPSIS

function result = ne(p1, p2 , varargin)

DESCRIPTION

NE overloads the ~= operator for pole objects.

%%%

DESCRIPTION: NE overloads the ~= operator for pole objects.

CALL: result = ne(p1,p1)
 result = ne(p1,p2, 'property1', 'property2')
 result = ne(p1,p2, 'class/property', 'class/property')

INPUTS: p1,p2 - input pole objects

OUTPUTS: If the two pole objects are considered equal, result == 0,
 otherwise, result == 1.

VERSION: \$Id: ne.m,v 1.2 2008/01/03 15:33:48 ingo Exp \$

HISTORY: 29-08-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a pole object into a string.
- [eq](#) EQ overloads the == operator for pole objects.

This function is called by:

SOURCE CODE

```

0001 function result = ne(p1, p2 , varargin)
0002 % NE overloads the ~= operator for pole objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for pole objects.
0007 %
0008 % CALL:          result = ne(p1,p1)
0009 %               result = ne(p1,p2, 'property1', 'property2')
0010 %               result = ne(p1,p2, 'class/property', 'class/property')
0011 %
0012 % INPUTS:        p1,p2 - input pole objects
0013 %
0014 % OUTPUTS:       If the two pole objects are considered equal, result == 0,
0015 %               otherwise, result == 1.
0016 %
0017 % VERSION:       $Id: ne.m,v 1.2 2008/01/03 15:33:48 ingo Exp $
0018 %
0019 % HISTORY:       29-08-2007 M Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: ne.m,v 1.2 2008/01/03 15:33:48 ingo Exp $';
0025
0026 % Check if this is a call for parameters
0027 if nargin == 2
0028     if isa(p1, 'pole') && ischar(p2)
0029         in = char(p2);
0030         if strcmp(in, 'Params')
0031             result = plist();
0032             return
0033         elseif strcmp(in, 'Version')
0034             result = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 if eq(p1, p2, varargin{:})
0041     result = 0;
0042 else
0043     result = 1;
0044 end
0045
0046 % END

```

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 1.7 2007/11/23 12:11:08 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for cdata objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for cdata objects.
0007 %
```

```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:        $Id: subsref.m,v 1.7 2007/11/23 12:11:08 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: subsref.m,v 1.7 2007/11/23 12:11:08 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(A, 'cdata') && ischar(S)
0022         in = char(S);
0023         if strcmp(in, 'Params')
0024             B = plist();
0025             return
0026         elseif strcmp(in, 'Version')
0027             B = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 B = builtin('subsref', A, S);
0034

```

[Home](#) > [m](#) > [noisegenerator](#) > Contents.m

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 22-Jan-2008 10:39:13 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

```
0001 function pth = get_curr_m_file_path (m_file_name)
0002 % GET_CURR_M_FILE_PATH returns the path for a mfile.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET_CURR_M_FILE_PATH returns the path for a mfile.
0007 %
0008 % CALL:            path = get_curr_m_file_path(mfilename)
```

```
0009 %           path = get_curr_m_file_path('explore_ao')
0010 %
0011 % VERSION:      $Id: get_curr_m_file_path.m,v 1.2 2007/06/25 11:21:48 ingo Exp $
0012 %
0013 % HISTORY: 25-06-07 Diepholz
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 pth = '';
0019 eval (sprintf('pth = which('%s');',m_file_name))
0020 index = find(pth==filesep, 1, 'last');
0021 pth    = pth(1:index);
```


[Home](#) > [classes](#) > [@provenance](#) > ne.m

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')

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 1.2 2008/01/07 13:48:02 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

```

0001 function result = ne(p1, p2, varargin)
0002 % NE overloads the ~= operator for provenance objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for provenance objects.
0007 %
0008 % CALL:      result = ne(p1,p2)
0009 %            result = ne(p1,p2, 'property1', 'property2')
0010 %            result = ne(p1,p2, 'class/property', 'class/property')
0011 %
0012 % INPUTS:    p1,p2 - input provenance objects
0013 %
0014 % OUTPUTS:   If the two provenance objects are considered equal, result == 0,
0015 %            otherwise, result == 1.
0016 %
0017 % VERSION:   $Id: ne.m,v 1.2 2008/01/07 13:48:02 ingo Exp $
0018 %
0019 % HISTORY:   29-08-2007 M Hewitson
0020 %            Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: ne.m,v 1.2 2008/01/07 13:48:02 ingo Exp $';
0025
0026 % Check if this is a call for parameters
0027 if nargin == 2
0028     if isa(p1, 'provenance') && ischar(p2)
0029         in = char(p2);
0030         if strcmp(in, 'Params')
0031             result = plist();
0032             return
0033         elseif strcmp(in, 'Version')
0034             result = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 if eq(p1, p2, varargin{:})
0041     result = 0;
0042 else
0043     result = 1;
0044 end
0045
0046 % END

```

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 1.13 2007/11/23 12:11:08 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for fsdata objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for fsdata objects.
0007 %
```

```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:        $Id: subsref.m,v 1.13 2007/11/23 12:11:08 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsref.m,v 1.13 2007/11/23 12:11:08 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'fsdata') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             B = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             B = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 B = builtin('subsref', A, S);

```

Contents

PURPOSE

%% path: m/noisegenerator/private %%%
%%

SYNOPSIS

This is a script file.

DESCRIPTION

%% path: m/noisegenerator/private %%%

 m/noisegenerator/
private/conv_noisegen - normalize to bn = 1
 m/noisegenerator/private/fq2fac - (No help available)

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > [get_window.m](#)

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 1.3 2007/05/30 05:19:55 hewitson 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\(N, psll\)](#)

SOURCE CODE

```
0001 function w = get_window(varargin)
0002
0003 % GET_WINDOW returns the required window function as a structure.
0004 %
```

```

0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: get_window.m,v 1.3 2007/05/30 05:19:55 hewitson Exp $
0008 %
0009
0010 PSLL = [];
0011
0012 % Get inputs
0013 if nargin == 2
0014     name = varargin{1};
0015     N     = varargin{2};
0016 elseif nargin == 3
0017     name = varargin{1};
0018     N     = varargin{2};
0019     PSLL = varargin{3};
0020 else
0021     error('### Incorrect inputs.');
```

```

0022 end
0023
0024 % Get window
0025 if isempty(PSLL)
0026     % get standard window
0027     z = [0:N-1]./N;
0028     eval(sprintf('w = win_%s(z);', name));
0029
0030 else
0031
0032     % Deal with Kaiser
0033     w = get_kaiser_window(N, PSLL);
0034
0035 end
0036
0037 %-----
0038 % Make Kaiser window
0039 %
0040 function w = get_kaiser_window(N, psll)
0041
0042 % Make window struct
0043 w.name      = 'Kaiser';
0044 w.alpha     = kaiser_alpha(psll);
0045 w.psll      = psll;
0046 w.rov       = kaiser_rov(w.alpha);
0047 w.nenbw     = kaiser_nenbw(w.alpha);
0048 w.w3db      = kaiser_w3db(w.alpha);
0049 w.flatness  = kaiser_flatness(w.alpha);
0050
0051 % Compute window samples
0052 % - here we adjust to make the window asymmetric
0053 v = kaiser(N+1, pi*w.alpha);
```

```
0054 v = v(1:end-1);  
0055 w.ws      = sum(v);  
0056 w.ws2     = sum(v.*v);  
0057 w.win     = v.';  
0058  
0059  
0060 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@pzmodel](#) > ne.m

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,p1)
 result = ne(p1,p2, exc_list)
 result = ne(p1,p2, 'property1', 'property2')
 result = ne(p1,p2, 'class/property', 'class/property')

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 1.3 2008/01/14 20:03:52 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

```

0001 function result = ne(p1, p2, varargin)
0002 % NE overloads the ~= operator for pzmodel objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for pzmodel objects.
0007 %
0008 % CALL:          result = ne(p1,p1)
0009 %               result = ne(p1,p2,  exc_list)
0010 %               result = ne(p1,p2, 'property1', 'property2')
0011 %               result = ne(p1,p2, 'class/property', 'class/property')
0012 %
0013 % INPUTS:        p1,p2 - input pzmodel objects
0014 %
0015 % OUTPUTS:       If the two pzmodel objects are considered equal, result == 0,
0016 %               otherwise, result == 1.
0017 %
0018 % VERSION:       $Id: ne.m,v 1.3 2008/01/14 20:03:52 ingo Exp $
0019 %
0020 % HISTORY:       29-08-2007 M Hewitson
0021 %               Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: ne.m,v 1.3 2008/01/14 20:03:52 ingo Exp $';
0026
0027 % Check if this is a call for parameters
0028 if nargin == 2
0029     if isa(p1, 'pzmodel') && ischar(p2)
0030         in = char(p2);
0031         if strcmp(in, 'Params')
0032             result = plist();
0033             return
0034         elseif strcmp(in, 'Version')
0035             result = VERSION;
0036             return
0037         end
0038     end
0039 end
0040
0041 if eq(p1, p2, varargin{:})
0042     result = 0;
0043 else
0044     result = 1;
0045 end
0046
0047 % END

```


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 1.3 2007/12/05 16:26:22 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for history objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for history objects.
0007 %
```

```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:        $Id: subsref.m,v 1.3 2007/12/05 16:26:22 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: subsref.m,v 1.3 2007/12/05 16:26:22 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(A, 'history') && ischar(S)
0022         in = char(S);
0023         if strcmp(in, 'Params')
0024             B = plist;
0025             return
0026         elseif strcmp(in, 'Version')
0027             B = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 B = builtin('subsref', A, S);
0034

```

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 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@cdata](#) > [get_xy_values.m](#)

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 1.3 2007/11/28 13:03:32 ingo 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

```
0001 function varargout = get_xy_values(data, pl)
0002 % GET_XY_VALUES returns the values in vals.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
```



```

0006 % DESCRIPTION: GET_XY_VALUES returns the values in vals.
0007 %
0008 % CALL:          y      = get_xy_values(ao.data);
0009 %                [x,y] = get_xy_values(ao.data);      --> x is an empty array
0010 %                [x,y] = get_xy_values(ao.data, pl);  --> the pl has no effect
0011 %
0012 % VERSION:       $Id: get_xy_values.m,v 1.3 2007/11/28 13:03:32 ingo Exp $
0013 %
0014 % HISTORY:       25-05-2007 Diepholz
0015 %                Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 VERSION = '$Id: get_xy_values.m,v 1.3 2007/11/28 13:03:32 ingo Exp $';
0020
0021 % Check if this is a call for parameters
0022 if nargin == 2
0023     if isa(data, 'cdata') && ischar(pl)
0024         in = char(pl);
0025         if strcmp(in, 'Params')
0026             varargout{1} = plist;
0027             return
0028         elseif strcmp(in, 'Version')
0029             varargout{1} = VERSION;
0030             return
0031         end
0032     end
0033 end
0034
0035 x = [];
0036 y = [];
0037
0038 %% Set output values
0039 y = data.y;
0040
0041 %% Set outputs
0042 if nargout == 1
0043     varargout{1} = y;
0044
0045 elseif nargout == 2
0046     varargout{1} = x;
0047     varargout{2} = y;
0048
0049 else
0050     error('### Incorrect output arguments.');
```

[Home](#) > [classes](#) > [@specwin](#) > ne.m

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')

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 1.2 2008/01/11 14:04:43 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

```

0001 function result = ne(w1, w2, varargin)
0002 % NE overloads the ~= operator for specwin objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for specwin objects.
0007 %
0008 % CALL:          result = ne(w1,w2)
0009 %               result = ne(w1,w2, 'property1', 'property2')
0010 %               result = ne(w1,w2, 'class/property', 'class/property')
0011 %
0012 % INPUTS:        w1,w2 - input specwin objects
0013 %
0014 % OUTPUTS:       If the two specwin objects are considered equal, result == 0,
0015 %               otherwise, result == 1.
0016 %
0017 % VERSION:       $Id: ne.m,v 1.2 2008/01/11 14:04:43 ingo Exp $
0018 %
0019 % HISTORY:       29-08-2007 M Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: ne.m,v 1.2 2008/01/11 14:04:43 ingo Exp $';
0025
0026 % Check if this is a call for parameters
0027 if nargin == 2
0028     if isa(w1, 'specwin') && ischar(w2)
0029         in = char(w2);
0030         if strcmp(in, 'Params')
0031             result = plist();
0032             return
0033         elseif strcmp(in, 'Version')
0034             result = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 if eq(w1, w2, varargin{:})
0041     result = 0;
0042 else
0043     result = 1;
0044 end
0045
0046 % END

```

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 1.2 2007/12/10 18:07:17 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for mfir objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for mfir objects.
0007 %
```

```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:        $Id: subsref.m,v 1.2 2007/12/10 18:07:17 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsref.m,v 1.2 2007/12/10 18:07:17 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'mfir') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             B = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             B = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 B = builtin('subsref', A, S);

```

[Home](#) > [m](#) > [sigproc](#) > [frequency_domain](#) > Contents.m

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 A0.  
    <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.

[m/sigproc/frequency_domain/ltpda_pwelch](matlab:help m/sigproc/frequency_domain/ltpda_pwelch) - makes power spectral density estimates of the time-series objects

[m/sigproc/frequency_domain/ltpda_rms](matlab:help m/sigproc/frequency_domain/ltpda_rms) - Calculate RMS deviation from spectrum

[m/sigproc/frequency_domain/ltpda_tfe](matlab:help m/sigproc/frequency_domain/ltpda_tfe) - makes transfer function estimates of the time-series objects.

[m/sigproc/frequency_domain/ltpda_xspec](matlab:help m/sigproc/frequency_domain/ltpda_xspec) - performs cross-spectral analysis of various forms.

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

Generated on Tue 22-Jan-2008 10:39:13 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 1.3 2007/11/28 13:03:32 ingo 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

```

0001 function varargout = get_xy_values(data, pl)
0002 % GET_XY_VALUES returns the x-axis and/or y-axis values.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET_XY_VALUES returns the x-axis and/or y-axis values.
0007 %
0008 % CALL:          x      = get_xy_values(ao.data);
0009 %                [x,y] = get_xy_values(ao.data);
0010 %                [x,y] = get_xy_values(ao.data, pl);
0011 %                y      = get_xy_values(ao.data, pl); --> only possible with
0012 %                                                         a parameter list
0013 %
0014 % REMARK:         In the case with TWO output arguments and only one parameter in
0015 %                 'pl' the function returns a empty array for the axis which is
0016 %                 not set in the parameterlist.
0017 %
0018 % PARAMETER LIST: <key>    <value>    <description>
0019 %                 'xdata'    'x'        returns the values of the x-axis
0020 %                 'ydata'    'y'        returns the values of the y-axis
0021 %
0022 % VERSION:        $Id: get_xy_values.m,v 1.3 2007/11/28 13:03:32 ingo Exp $
0023 %
0024 % HISTORY:        25-05-2007 Diepholz
0025 %                 Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 VERSION = '$Id: get_xy_values.m,v 1.3 2007/11/28 13:03:32 ingo Exp $';
0030
0031 % Check if this is a call for parameters
0032 if nargin == 2
0033     if isa(data, 'fsdata') && ischar(pl)
0034         in = char(pl);
0035         if strcmp(in, 'Params')
0036             varargout{1} = plist;
0037             return
0038         elseif strcmp(in, 'Version')
0039             varargout{1} = VERSION;
0040             return
0041         end
0042     end
0043 end
0044
0045 if nargin < 2
0046     pl = [];
0047 end

```

```

0048
0049 x  = [];
0050 y  = [];
0051
0052 %% Set default parameter list
0053 if nargout == 1
0054     pl_default = plist(param('xdata', 'x'));
0055 else %if nargout == 2
0056     pl_default = plist([param('xdata', 'x')
0057                         param('ydata', 'y')]);
0058 end
0059
0060
0061 %% Combine the handover param-list and the default param-list
0062 if ~isempty(pl)
0063     pl = combine(pl);
0064 else
0065     pl = combine(pl_default);
0066 end
0067
0068 do_xdata = find(pl, 'xdata');
0069 do_ydata = find(pl, 'ydata');
0070
0071 %% Make sure that at least one axis is defined
0072 if isempty(do_xdata) && isempty(do_ydata)
0073     do_xdata = 'x';
0074 end
0075
0076 %% Set x-axis and y-axis values
0077 if ~isempty(do_xdata)
0078     x = data.x;
0079 end
0080
0081 if ~isempty(do_ydata)
0082     y = data.y;
0083 end
0084
0085 %% Set outputs
0086 if nargout == 1
0087     if ~isempty(x)
0088         varargout{1} = x;
0089     else % ~isempty(y)
0090         varargout{1} = y;
0091     end
0092
0093 elseif nargout == 2
0094     varargout{1} = x;
0095     varargout{2} = y;
0096
0097 else
0098     error('### Incorrect output arguments.');
```

```
0100
```

```
0101 % END
```

Generated on Tue 22-Jan-2008 10:39:13 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')

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 1.2 2008/01/14 20:03:52 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

```

0001 function result = ne(t1, t2, varargin)
0002 % NE overloads the ~= operator for time objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for time objects.
0007 %
0008 % CALL:          result = ne(t1,t2)
0009 %               result = ne(t1,t2,  exc_list)
0010 %               result = ne(t1,t2, 'property1', 'property2')
0011 %               result = ne(t1,t2, 'class/property', 'class/property')
0012 %
0013 % INPUTS:        t1,t2 - input time objects
0014 %
0015 % OUTPUTS:       If the two time objects are considered equal, result == 0,
0016 %               otherwise, result == 1.
0017 %
0018 % VERSION:       $Id: ne.m,v 1.2 2008/01/14 20:03:52 ingo Exp $
0019 %
0020 % HISTORY:       29-08-2007 M Hewitson
0021 %               Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: ne.m,v 1.2 2008/01/14 20:03:52 ingo Exp $';
0026
0027 % Check if this is a call for parameters or for the cvs-version number
0028 if nargin == 2
0029     if isa(t1, 'time') && ischar(t2)
0030         in = char(t2);
0031         if strcmp(in, 'Params')
0032             result = plist();
0033             return
0034         elseif strcmp(in, 'Version')
0035             result = VERSION;
0036             return
0037         end
0038     end
0039 end
0040
0041 if eq(t1, t2, varargin{:})
0042     result = 0;
0043 else
0044     result = 1;
0045 end
0046
0047 % END

```


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 1.2 2007/12/10 18:07:17 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for miir objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for miir objects.
0007 %
```

```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:        $Id: subsref.m,v 1.2 2007/12/10 18:07:17 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsref.m,v 1.2 2007/12/10 18:07:17 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'miir') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             B = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             B = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 B = builtin('subsref', A, S);

```


[Home](#) > [m](#) > [sigproc](#) > [time_domain](#) > Contents.m

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 A0(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 A0s to
    a target time-series A0.
    <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 22-Jan-2008 10:39:13 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 1.3 2007/11/28 13:03:32 ingo 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

```

0001 function varargout = get_xy_values(data, pl)
0002 % GET_XY_VALUES returns the x-axis and/or y-axis values.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET_XY_VALUES returns the x-axis and/or y-axis values.
0007 %
0008 % CALL:          x      = get_xy_values(ao.data);
0009 %                [x,y] = get_xy_values(ao.data);
0010 %                [x,y] = get_xy_values(ao.data, pl);
0011 %                y      = get_xy_values(ao.data, pl); --> only possible with
0012 %                                                         a parameter list
0013 %
0014 % REMARK:         In the case with TWO output arguments and only one parameter in
0015 %                 'pl' the function returns a empty array for the axis which is
0016 %                 not set in the parameterlist.
0017 %
0018 % PARAMETER LIST: <key>   <value>   <description>
0019 %                 'xdata'   'x'       returns the values of the x-axis
0020 %                 'ydata'   'y'       returns the values of the y-axis
0021 %
0022 % VERSION:        $Id: get_xy_values.m,v 1.3 2007/11/28 13:03:32 ingo Exp $
0023 %
0024 % HISTORY:        25-05-2007 Diepholz
0025 %                 Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 VERSION = '$Id: get_xy_values.m,v 1.3 2007/11/28 13:03:32 ingo Exp $';
0030
0031 % Check if this is a call for parameters
0032 if nargin == 2
0033     if isa(data, 'tsdata') && ischar(pl)
0034         in = char(pl);
0035         if strcmp(in, 'Params')
0036             varargout{1} = plist;
0037             return
0038         elseif strcmp(in, 'Version')
0039             varargout{1} = VERSION;
0040             return
0041         end
0042     end
0043 end
0044
0045 if nargin < 2
0046     pl = [];
0047 end

```

```

0048
0049 x  = [];
0050 y  = [];
0051
0052 %% Set default parameter list
0053 if nargout == 1
0054     pl_default = plist(param('xdata', 'x'));
0055 else %if nargout == 2
0056     pl_default = plist([param('xdata', 'x')
0057                         param('ydata', 'y')]);
0058 end
0059
0060
0061 %% Combine the handover param-list and the default param-list
0062 if ~isempty(pl)
0063     pl = combine(pl);
0064 else
0065     pl = combine(pl_default);
0066 end
0067
0068 do_xdata = find(pl, 'xdata');
0069 do_ydata = find(pl, 'ydata');
0070
0071 %% Make sure that at least one axis is defined
0072 if isempty(do_xdata) && isempty(do_ydata)
0073     do_xdata = 'x';
0074 end
0075
0076 %% Set x-axis and y-axis values
0077 if ~isempty(do_xdata)
0078     x = data.x;
0079 end
0080
0081 if ~isempty(do_ydata)
0082     y = data.y;
0083 end
0084
0085 %% Set outputs
0086 if nargout == 1
0087     if ~isempty(y)
0088         varargout{1} = x;
0089     else % ~isempty(y)
0090         varargout{1} = y;
0091     end
0092
0093 elseif nargout == 2
0094     varargout{1} = x;
0095     varargout{2} = y;
0096
0097 else
0098     error('### Incorrect output arguments.');
```

```
0100
```

```
0101 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@timeformat](#) > ne.m

ne

PURPOSE

NE overloads the ~= operator for timeformat objects.

SYNOPSIS

function result = ne(tf1, tf2, 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')

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 1.2 2008/01/15 17:36:36 ingo 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

```

0001 function result = ne(tf1, tf2, varargin)
0002 % NE overloads the ~= operator for timeformat objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for timeformat objects.
0007 %
0008 % CALL:          result = ne(tf1,tf2)
0009 %               result = ne(tf1,tf2,  exc_list)
0010 %               result = ne(tf1,tf2, 'property1', 'property2')
0011 %               result = ne(tf1,tf2, 'class/property', 'class/property')
0012 %
0013 % INPUTS:        tf1,tf2 - input timeformat objects
0014 %
0015 % OUTPUTS:       If the two timeformat objects are considered equal, result == 0,
0016 %               otherwise, result == 1.
0017 %
0018 % VERSION:       $Id: ne.m,v 1.2 2008/01/15 17:36:36 ingo Exp $
0019 %
0020 % HISTORY:       29-08-2007 M Hewitson
0021 %               Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: ne.m,v 1.2 2008/01/15 17:36:36 ingo Exp $';
0026
0027 % Check if this is a call for parameters
0028 if nargin == 2
0029     if isa(tf1, 'timeformat') && ischar(tf2)
0030         in = char(tf2);
0031         if strcmp(in, 'Params')
0032             result = plist();
0033             return
0034         elseif strcmp(in, 'Version')
0035             result = VERSION;
0036             return
0037         end
0038     end
0039 end
0040
0041 if eq(tf1, tf2, varargin{:})
0042     result = 0;
0043 else
0044     result = 1;
0045 end
0046
0047 % END

```


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 1.7 2007/12/11 17:27:09 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for parameter objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for parameter objects.
0007 %
```

```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:        $Id: subsref.m,v 1.7 2007/12/11 17:27:09 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsref.m,v 1.7 2007/12/11 17:27:09 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'param') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             B = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             B = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 B = builtin('subsref', A, S);
0033

```

[Home](#) > [m](#) > [timetools](#) > Contents.m

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 22-Jan-2008 10:39:13 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 1.2 2007/11/28 13:03:32 ingo 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

```

0001 function varargout = get_xy_values(data, pl)
0002 % GET_XY_VALUES returns the x-axis and/or y-axis values.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET_XY_VALUES returns the x-axis and/or y-axis values.
0007 %
0008 % CALL:          x      = get_xy_values(ao.data);
0009 %               [x,y] = get_xy_values(ao.data);
0010 %               [x,y] = get_xy_values(ao.data, pl);
0011 %               y      = get_xy_values(ao.data, pl); --> only possible with
0012 %                                                           a parameter list
0013 %
0014 % REMARK:          In the case with TWO output arguments and only one parameter in
0015 %                  'pl' the function returns a empty array for the axis which is
0016 %                  not set in the parameterlist.
0017 %
0018 % PARAMETER LIST:  <key>    <value>    <description>
0019 %                  'xdata'    'x'        returns the values of the x-axis
0020 %                  'ydata'    'y'        returns the values of the y-axis
0021 %
0022 % VERSION:          $Id: get_xy_values.m,v 1.2 2007/11/28 13:03:32 ingo Exp $
0023 %
0024 % HISTORY:          25-05-2007 Diepholz
0025 %                  Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 VERSION = '$Id: get_xy_values.m,v 1.2 2007/11/28 13:03:32 ingo Exp $';
0030
0031 % Check if this is a call for parameters
0032 if nargin == 2
0033     if isa(data, 'xydata') && ischar(pl)
0034         in = char(pl);
0035         if strcmp(in, 'Params')
0036             varargout{1} = plist;
0037             return
0038         elseif strcmp(in, 'Version')
0039             varargout{1} = VERSION;
0040             return
0041         end
0042     end
0043 end
0044
0045 if nargin < 2
0046     pl = [];
0047 end

```

```

0048
0049 x  = [];
0050 y  = [];
0051
0052 %% Set default parameter list
0053 if nargout == 1
0054     pl_default = plist(param('xdata', 'x'));
0055 else %if nargout == 2
0056     pl_default = plist([param('xdata', 'x')
0057                        param('ydata', 'y')]);
0058 end
0059
0060
0061 %% Combine the handover param-list and the default param-list
0062 if ~isempty(pl)
0063     pl = combine(pl);
0064 else
0065     pl = combine(pl_default);
0066 end
0067
0068 do_xdata = find(pl, 'xdata');
0069 do_ydata = find(pl, 'ydata');
0070
0071 %% Make sure that at least one axis is defined
0072 if isempty(do_xdata) && isempty(do_ydata)
0073     do_xdata = 'x';
0074 end
0075
0076 %% Set x-axis and y-axis values
0077 if ~isempty(do_xdata)
0078     x = data.x;
0079 end
0080
0081 if ~isempty(do_ydata)
0082     y = data.y;
0083 end
0084
0085 %% Set outputs
0086 if nargout == 1
0087     if ~isempty(x)
0088         varargout{1} = x;
0089     else % ~isempty(y)
0090         varargout{1} = y;
0091     end
0092
0093 elseif nargout == 2
0094     varargout{1} = x;
0095     varargout{2} = y;
0096
0097 else
0098     error('### Incorrect output arguments. ');
0099 end

```

```
0100
```

```
0101 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@timespan](#) > ne.m

ne

PURPOSE

NE overloads the ~= operator for timespan objects.

SYNOPSIS

function result = ne(t1,t2)

DESCRIPTION

NE overloads the ~= operator for timespan objects.

%%%

DESCRIPTION: NE overloads the ~= operator for timespan objects.

CALL: result = ne(ts1,ts2)

INPUTS: t1,t2 - input timespan objects

OUTPUTS: If the two timespan objects are considered equal, result == 0,
 otherwise, result == 1.

VERSION: \$Id: ne.m,v 1.1 2007/08/29 18:41:27 hewitson Exp \$

HISTORY: 29-08-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [eq](#) EQ overloads the == operator for timespan objects.

This function is called by:

SOURCE CODE

```
0001 function result = ne(t1,t2)
0002
```



```

0003 % NE overloads the ~= operator for timespan objects.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: NE overloads the ~= operator for timespan objects.
0008 %
0009 % CALL:          result = ne(ts1,ts2)
0010 %
0011 % INPUTS:        t1,t2 - input timespan objects
0012 %
0013 % OUTPUTS:        If the two timespan objects are considered equal, result == 0,
0014 %                  otherwise, result == 1.
0015 %
0016 % VERSION:        $Id: ne.m,v 1.1 2007/08/29 18:41:27 hewitson Exp $
0017 %
0018 % HISTORY:        29-08-2007 M Hewitson
0019 %                  Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 if eq(t1,t2)
0024     result = 0;
0025 else
0026     result = 1;
0027 end
0028
0029 % END

```

[Home](#) > [classes](#) > [@pzmodel](#) > [private](#) > abccascade.m

abccascade

PURPOSE

Cascade two filters together to get a new filter.

SYNOPSIS

```
function [ao,bo] = abccascade(a1,b1,a2,b2)
```

DESCRIPTION

Cascade two filters together to get a new filter.

usage: [a,b] = abccascade(a1,b1,a2,b2)

M Hewitson 03-04-07

\$Id: abccascade.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

```
0001 function [ao,bo] = abccascade(a1,b1,a2,b2)
0002
0003 % Cascade two filters together to get a new filter.
0004 %
```

```
0005 % usage: [a,b] = abcascade(a1,b1,a2,b2)
0006 %
0007 % M Hewitson 03-04-07
0008 %
0009 % $Id: abcascade.m,v 1.1 2007/04/03 11:34:23 hewitson Exp $
0010 %
0011
0012 n = length(a1);
0013 m = length(a2);
0014 N = m+n-1;
0015
0016 alpha = zeros(1, N);
0017 beta = zeros(1, N);
0018
0019 for i=0:n-1
0020     for j=0:m-1
0021         alpha(i+j+1) = alpha(i+j+1) + a1(i+1)*a2(j+1);
0022         beta(i+j+1) = beta(i+j+1) + b1(i+1)*b2(j+1);
0023     end
0024 end
0025
0026 ao = alpha;
0027 bo = beta;
```

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 1.1 2007/12/24 19:47:06 hewitson Exp \$

HISTORY: 24-12-2007 Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```

0001 function varargout = get_xy_values(data, pl)
0002 % GET_XYZ_VALUES returns the x-axis and/or y-axis and/or z-axis values.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET_XYZ_VALUES returns the x-axis and/or y-axis and/or z-axis
0007 %                values.
0008 %
0009 % CALL:          x      = get_xyz_values(ao.data);
0010 %                [x,y] = get_xyz_values(ao.data);
0011 %                [x,y,z] = get_xyz_values(ao.data);
0012 %                [x,y] = get_xyz_values(ao.data, pl);
0013 %                [x,y,z] = get_xyz_values(ao.data, pl);
0014 %                z      = get_xyz_values(ao.data, pl); --> only possible with
0015 %                                                           a parameter list
0016 %
0017 % REMARK:         In the case with 3 output arguments and only one parameter in
0018 %                  'pl' the function returns a empty array for the axis which is
0019 %                  not set in the parameterlist.
0020 %
0021 % PARAMETER LIST: <key>    <value>    <description>
0022 %                  'xdata'    'x'      returns the values of the x-axis
0023 %                  'ydata'    'y'      returns the values of the y-axis
0024 %                  'zdata'    'z'      returns the values of the z-axis
0025 %
0026 % VERSION:        $Id: get_xyz_values.m,v 1.1 2007/12/24 19:47:06 hewitson Exp
0027 % $
0028 % HISTORY:         24-12-2007 Hewitson
0029 %                  Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032 %
0033 VERSION = '$Id: get_xyz_values.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $';
0034 %
0035 % Check if this is a call for parameters
0036 if nargin == 2
0037     if ischar(pl)
0038         in = char(pl);
0039         if strcmp(in, 'Params')
0040             varargout{1} = plist;
0041             return
0042         elseif strcmp(in, 'Version')
0043             varargout{1} = VERSION;
0044             return
0045         end
0046     end

```

```

0047 end
0048
0049 if nargin < 2
0050     pl = [];
0051 end
0052
0053 x = [];
0054 y = [];
0055
0056 %% Set default parameter list
0057 if nargin == 1
0058     pl_default = plist(param('xdata', 'x'));
0059 elseif nargin == 2
0060     pl_default = plist([param('xdata', 'x')
0061                         param('ydata', 'y')]);
0062 elseif nargin == 3
0063     pl_default = plist('xdata', 'x', 'ydata', 'y', 'zdata', 'z');
0064 else
0065     error('### Incorrect outputs. ');
0066 end
0067
0068
0069 %% Combine the handover param-list and the default param-list
0070 if ~isempty(pl)
0071     pl = combine(pl);
0072 else
0073     pl = combine(pl_default);
0074 end
0075
0076 do_xdata = find(pl, 'xdata');
0077 do_ydata = find(pl, 'ydata');
0078 do_zdata = find(pl, 'zdata');
0079
0080 %% Make sure that at least one axis is defined
0081 if isempty(do_xdata) && isempty(do_ydata)
0082     do_xdata = 'x';
0083 end
0084
0085 %% Set x-axis and y-axis values
0086 if ~isempty(do_xdata)
0087     x = data.x;
0088 end
0089
0090 if ~isempty(do_ydata)
0091     y = data.y;
0092 end
0093
0094 if ~isempty(do_zdata)
0095     z = data.z;
0096 end
0097
0098 %% Set outputs
0099 if nargin == 1

```

```
0100     if ~isempty(x)
0101         varargout{1} = x;
0102     else % ~isempty(y)
0103         varargout{1} = y;
0104     end
0105
0106 elseif nargout == 2
0107     varargout{1} = x;
0108     varargout{2} = y;
0109
0110 elseif nargout == 3
0111     varargout{1} = x;
0112     varargout{2} = y;
0113     varargout{3} = z;
0114 else
0115     error('### Incorrect output arguments. ');
0116 end
0117
0118 % END
```

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')

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 1.2 2007/11/28 13:03:32 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


```

0001 function result = ne(c1, c2, varargin)
0002 % NE overloads the ~= operator for tsdata objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for tsdata objects.
0007 %
0008 % CALL:          result = ne(c1,c2)
0009 %               result = ne(c1,c2, 'property1', 'property2')
0010 %               result = ne(c1,c2, 'class/property', 'class/property')
0011 %
0012 % INPUTS:        c1,c2 - input tsdata objects
0013 %
0014 % OUTPUTS:       If the two tsdata objects are considered equal, result == 0,
0015 %               otherwise, result == 1.
0016 %
0017 % VERSION:       $Id: ne.m,v 1.2 2007/11/28 13:03:32 ingo Exp $
0018 %
0019 % HISTORY:       29-08-2007 M Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: ne.m,v 1.2 2007/11/28 13:03:32 ingo Exp $';
0025
0026 % Check if this is a call for parameters
0027 if nargin == 2
0028     if isa(c1, 'tsdata') && ischar(c2)
0029         in = char(c2);
0030         if strcmp(in, 'Params')
0031             result = plist;
0032             return
0033         elseif strcmp(in, 'Version')
0034             result = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 if eq(c1,c2, varargin{:})
0041     result = 0;
0042 else
0043     result = 1;
0044 end
0045
0046 % END

```

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 1.2 2008/01/03 15:33:48 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for pole objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for pole objects.
0007 %
```

```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:        $Id: subsref.m,v 1.2 2008/01/03 15:33:48 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsref.m,v 1.2 2008/01/03 15:33:48 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'pole') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             B = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             B = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 B = builtin('subsref', A, S);

```

[Home](#) > [classes](#) > [@pzmodel](#) > getlowerFreq.m

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 1.3 2008/01/07 18:13:10 ingo 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

```
0001 function f = getlowerFreq(pzm, varargin)
0002 % GETLOWERFREQ gets the frequency of the lowest pole or zero in the model.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GETLOWERFREQ gets the frequency of the lowest pole or zero in the
0007 %               model.
0008 %
0009 % CALL:         f = getlowerFreq(pzm);
0010 %
0011 % VERSION:      $Id: getlowerFreq.m,v 1.3 2008/01/07 18:13:10 ingo Exp $
0012 %
0013 % HISTORY:      04-04-2007 M Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: getlowerFreq.m,v 1.3 2008/01/07 18:13:10 ingo Exp $';
0019
0020 % Check if this is a call for parameters
0021 if nargin == 2
0022     if isa(pzm, 'pzmodel') && ischar(varargin{1})
0023         in = char(varargin{1});
0024         if strcmp(in, 'Params')
0025             f = plist;
0026             return
0027         elseif strcmp(in, 'Version')
0028             f = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 poles = get(pzm, 'poles');
0035 zeros = get(pzm, 'zeros');
0036 np = length(poles);
0037 nz = length(zeros);
0038 f = 1e10;
0039 for j=1:np
0040     pole = poles(j);
0041     fc = get(pole, 'f');
0042     if fc < f
0043         f = fc;
0044     end
0045 end
0046
0047 for j=1:nz
0048     zero = zeros(j);
0049     fc = get(zero, 'f');
0050     if fc < f
0051         f = fc;
0052     end
0053 end
0054
0055

```

0056

0057 % END

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@xydata](#) > ne.m

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')

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 1.2 2007/11/28 13:03:32 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

```

0001 function result = ne(c1, c2, varargin)
0002 % NE overloads the ~= operator for xydata objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for xydata objects.
0007 %
0008 % CALL:          result = ne(c1,c2)
0009 %               result = ne(c1,c2, 'property1', 'property2')
0010 %               result = ne(c1,c2, 'class/property', 'class/property')
0011 %
0012 % INPUTS:        c1,c2 - input xydata objects
0013 %
0014 % OUTPUTS:       If the two xydata objects are considered equal, result == 0,
0015 %               otherwise, result == 1.
0016 %
0017 % VERSION:       $Id: ne.m,v 1.2 2007/11/28 13:03:32 ingo Exp $
0018 %
0019 % HISTORY:       29-08-2007 M Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: ne.m,v 1.2 2007/11/28 13:03:32 ingo Exp $';
0025
0026 % Check if this is a call for parameters
0027 if nargin == 2
0028     if isa(c1, 'xydata') && ischar(c2)
0029         in = char(c2);
0030         if strcmp(in, 'Params')
0031             result = plist;
0032             return
0033         elseif strcmp(in, 'Version')
0034             result = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 if eq(c1,c2, varargin{:})
0041     result = 0;
0042 else
0043     result = 1;
0044 end
0045
0046 % END

```


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 1.3 2008/01/07 13:48:02 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for provenance objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for provenance objects.
0007 %
```

```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:       $Id: subsref.m,v 1.3 2008/01/07 13:48:02 ingo Exp $
0011 %
0012 % HISTORY:       31-01-07 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsref.m,v 1.3 2008/01/07 13:48:02 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'provenance') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             B = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             B = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 B = builtin('subsref', A, S);
0033

```

[Home](#) > [classes](#) > [@pzmodel](#) > getupperFreq.m

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 1.3 2008/01/07 18:13:10 ingo 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

```
0001 function f = getupperFreq(pzm, varargin)
0002 % GETUPPERFREQ gets the frequency of the highest pole or zero in the model.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GETUPPERFREQ gets the frequency of the highest pole or zero in
0007 %               the model.
0008 %
0009 % CALL:         f = getupperFreq(pzm);
0010 %
0011 % VERSION:      $Id: getupperFreq.m,v 1.3 2008/01/07 18:13:10 ingo Exp $
0012 %
0013 % HISTORY:      04-04-2007 M Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: getupperFreq.m,v 1.3 2008/01/07 18:13:10 ingo Exp $';
0019
0020 % Check if this is a call for parameters
0021 if nargin == 2
0022     if isa(pzm, 'pzmodel') && ischar(varargin{1})
0023         in = char(varargin{1});
0024         if strcmp(in, 'Params')
0025             f = plist;
0026             return
0027         elseif strcmp(in, 'Version')
0028             f = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 poles = get(pzm, 'poles');
0035 zeros = get(pzm, 'zeros');
0036 np = length(poles);
0037 nz = length(zeros);
0038 f = 0;
0039 for j=1:np
0040     pole = poles(j);
0041     fc = get(pole, 'f');
0042     if fc > f
0043         f = fc;
0044     end
0045 end
0046
0047 for j=1:nz
0048     zero = zeros(j);
0049     fc = get(zero, 'f');
0050     if fc > f
0051         f = fc;
0052     end
0053 end
0054
0055

```

0056

0057 % END

Generated on Tue 22-Jan-2008 10:39:13 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')

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 1.1 2007/12/24 19:47:06 hewitson 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

```

0001 function result = ne(c1, c2, varargin)
0002 % NE overloads the ~= operator for xyzdata objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for xyzdata objects.
0007 %
0008 % CALL:          result = ne(c1,c2)
0009 %               result = ne(c1,c2, 'property1', 'property2')
0010 %               result = ne(c1,c2, 'class/property', 'class/property')
0011 %
0012 % INPUTS:        c1,c2 - input xyzdata objects
0013 %
0014 % OUTPUTS:       If the two xyzdata objects are considered equal, result == 0,
0015 %               otherwise, result == 1.
0016 %
0017 % VERSION:       $Id: ne.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $
0018 %
0019 % HISTORY:       24-12-2007 M Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: ne.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $';
0025
0026 % Check if this is a call for parameters
0027 if nargin == 2
0028     if ischar(c2)
0029         in = char(c2);
0030         if strcmp(in, 'Params')
0031             result = plist;
0032             return
0033         elseif strcmp(in, 'Version')
0034             result = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 if eq(c1,c2, varargin{:})
0041     result = 0;
0042 else
0043     result = 1;
0044 end
0045
0046 % END

```

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 1.2 2008/01/08 14:57:07 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for pole/zero model objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for pole/zero model objects.
0007 %
```



```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:       $Id: subsref.m,v 1.2 2008/01/08 14:57:07 ingo Exp $
0011 %
0012 % HISTORY:       31-01-07 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsref.m,v 1.2 2008/01/08 14:57:07 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'pzmodel') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             B = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             B = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 B = builtin('subsref', A, S);

```

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')

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 1.2 2008/01/03 15:33:49 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

```

0001 function result = ne(p1, p2, varargin)
0002 % NE overloads the ~= operator for zero objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for zero objects.
0007 %
0008 % CALL:          result = ne(z1,z1)
0009 %               result = ne(p1,p2, 'property1', 'property2')
0010 %               result = ne(p1,p2, 'class/property', 'class/property')
0011 %
0012 % INPUTS:        z1,z2 - input zero objects
0013 %
0014 % OUTPUTS:       If the two zero objects are considered equal, result == 0,
0015 %               otherwise, result == 1.
0016 %
0017 % VERSION:       $Id: ne.m,v 1.2 2008/01/03 15:33:49 ingo Exp $
0018 %
0019 % HISTORY:       29-08-2007 M Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: ne.m,v 1.2 2008/01/03 15:33:49 ingo Exp $';
0025
0026 % Check if this is a call for parameters
0027 if nargin == 2
0028     if isa(p1, 'zero') && ischar(p2)
0029         in = char(p2);
0030         if strcmp(in, 'Params')
0031             result = plist();
0032             return
0033         elseif strcmp(in, 'Version')
0034             result = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 if eq(p1, p2, varargin{:})
0041     result = 0;
0042 else
0043     result = 1;
0044 end
0045
0046 % END

```

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 1.3 2008/01/11 14:04:44 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for specwin objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for specwin objects.
0007 %
```

```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:        $Id: subsref.m,v 1.3 2008/01/11 14:04:44 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsref.m,v 1.3 2008/01/11 14:04:44 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'specwin') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             B = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             B = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 B = builtin('subsref', A, S);
0033

```

[Home](#) > [m](#) > [noisegenerator](#) > ngconv.m

ngconv

PURPOSE

```
% get zeros and poles from pzmodels
```

SYNOPSIS

```
function [a,b] = ngconv(pzm)
```

DESCRIPTION

```
% get zeros and poles from pzmodels
```

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

```
0001 function [a,b] = ngconv(pzm)
0002
0003 %% get zeros and poles from pzmodels
0004
0005 zs = get(pzm, 'zeros');
0006 ps = get(pzm, 'poles');
0007
0008 %%
0009 f_zer = [];
0010 q_zer = [];
0011 for j=1:length(zs)
```

```
0012     z = zs(j);
0013     f_zer(j,1) = [get(z, 'f')];
0014     q_zer(j,1) = [get(z, 'q')];
0015     %if isnan(q_zer(j))
0016     %     q_zer(j,1) = 0;
0017     %end
0018     %zv(j,1:2) = [f q];
0019 end
0020
0021 f_pol = [];
0022 q_pol = [];
0023 for j=1:length(ps)
0024     p = ps(j);
0025     f_pol(j,1) = [get(p, 'f')];
0026     q_pol(j,1) = [get(p, 'q')];
0027     %if isnan(q_pol(j))
0028     %     q_pol(j,1) = 0;
0029     %end
0030 end
0031 %% calculate factors from f and q
0032 pol = fq2fac(f_pol,q_pol);
0033 zer = fq2fac(f_zer,q_zer);
0034 %%
0035 [b,a] = conv_noisegen(pol,zer);
0036
0037
0038
0039 %END
```

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 1.7 2008/01/15 17:36:36 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for time objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for time objects.
0007 %
```



```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:        $Id: subsref.m,v 1.7 2008/01/15 17:36:36 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsref.m,v 1.7 2008/01/15 17:36:36 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'time') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             B = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             B = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 B = builtin('subsref', A, S);
0033

```

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.

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

DESCRIPTION: FOO overloads the foo function for Analysis Objects.

CALL:
      foo(a)
      [a, b] = foo(a,pl)

INPUTS:
      a - analysis object(s)
      pl - parameter list(s)
      ...

OUTPUTS:
      a - ???
      b - ???

PARAMETERS:
      N: - description (Add this parameters to the default parameter list)
      X: - description

VERSION:
      $Id: ao_fcn_template.m,v 1.7 2008/01/15 06:24:51 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:

>> pl = foo(ao,'Version')
```

%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SUBFUNCTIONS

- `function plo = getDefaultPL()`

SOURCE CODE

```

0001 function varargout = foo(varargin)
0002 % F00 overloads the foo function for Analysis Objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: F00 overloads the foo function for Analysis Objects.
0007 %
0008 % CALL:
0009 %           foo(a)
0010 %           [a, b] = foo(a,pl)
0011 % INPUTS:
0012 %           a - analysis object(s)
0013 %           pl - parameter list(s)
0014 %           ...
0015 % OUTPUTS:
0016 %           a - ???
0017 %           b - ???
0018 % PARAMETERS: N: - description (Add this parameters to the default parameter
0019 %               list)
0020 %               X: - description
0021 % VERSION:     $Id: ao_fcn_template.m,v 1.7 2008/01/15 06:24:51 mauro Exp $
0022 %
0023 % HISTORY:     dd-mm-yyyy M Hewitson
0024 %               Creation
0025 %
0026 % The following call returns a parameter list object that contains the
0027 % default parameter values:
0028 %
0029 % >> pl = foo(ao,'Params')
0030 %
0031 % The following call returns a string that contains the routine CVS version:
0032 %
0033 % >> pl = foo(ao,'Version')
0034 %
0035 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0036 %
0037 ALGONAME = mfilename;

```

```

0038 VERSION = '$Id: ao_fcn_template.m,v 1.7 2008/01/15 06:24:51 mauro Exp $';
0039 bs       = [];
0040
0041 %% Check if this is a call for parameters or the cvs version string
0042 if nargin == 2
0043     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0044         in = char(varargin{2});
0045         if strcmpi(in, 'Params')
0046             varargout{1} = getDefaultPL();
0047             return
0048         elseif strcmpi(in, 'Version')
0049             varargout{1} = VERSION;
0050             return
0051         end
0052     end
0053 end
0054
0055 %% Collect input ao's, plist's and ao variable names
0056 in_names = {};
0057 for ii = 1:nargin
0058     in_names{end+1} = inputname(ii);
0059 end
0060
0061 [as, pl, invars] = collect_inputs(varargin, in_names);
0062
0063 % produce one parameter list
0064 if isa(pl, 'plist')
0065     pl = combine(pl, getDefaultPL());
0066 else
0067     pl = getDefaultPL();
0068 end
0069
0070 %% Get parameters
0071 N = find(pl, 'N');
0072 X = find(pl, 'X');
0073
0074 %% go through analysis objects
0075 for j=1: numel(as)
0076     a = as(j);
0077
0078     % get data
0079     [x,y] = get_xy_values(a.data);
0080
0081     % Do some analysis
0082     y = y;
0083     x = 1:length(y);
0084
0085     %%----- Build output AOs
0086
0087     % make a new xydata object, for example
0088     xy = xydata(x, y);
0089     xy = set(xy, 'name', sprintf('foo(%s)', a.data.name));

```

```

0090 xy = set(xy, 'xunits', a.data.yunits);
0091 xy = set(xy, 'yunits', 'Blah');
0092
0093 % make a new history object
0094 h = history(ALGONAME, VERSION, pl, a.hist);
0095 h = set(h, 'invars', invars);
0096
0097 % make output analysis object
0098 b = ao(xy, h);
0099
0100 % name for this object
0101 b = setnh(b, 'name', sprintf('foo(%s)', invars{j}));
0102 bs = [bs b];
0103
0104 end
0105
0106 %% Reshape the ouput to the same size of the input
0107 varargout{1} = reshape(bs, size(as));
0108
0109 %% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0110 %
0111 % FUNCTION:      getDefaultPL
0112 %
0113 % DESCRIPTION:  Get default params
0114 %
0115 % HISTORY:      dd-mm-yyyy M Hewitson
0116 %              Creation
0117 %
0118 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0119 function plo = getDefaultPL()
0120
0121 plo = plist('N', 10, ...
0122            'X', []);
0123
0124
0125 % END

```

[Home](#) > [classes](#) > [@history](#) > hist2dot.m

hist2dot

PURPOSE

HIST2DOT converts a history object to a 'DOT' file suitable for processing with graphviz

SYNOPSIS

function hist2dot(h, filename)

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 program graphviz

VERSION: \$Id: hist2dot.m,v 1.3 2007/07/18 13:58:44 ingo Exp \$

HISTORY: 23-05-2007 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a param object into a string.
- [getNode](#) GETNODES converts a history object to a nodes structure suitable for plotting as a tree.

This function is called by:

SOURCE CODE

```

0001 function hist2dot(h, filename)
0002 % HIST2DOT converts a history object to a 'DOT' file suitable for processing
with graphviz
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: HIST2DOT converts a history object to a 'DOT' file suitable for
0007 %               processing with graphviz (www.graphviz.org).
0008 %
0009 % CALL:         hist2dot(h, 'foo.dot');
0010 %
0011 % INPUT:        h          - history object
0012 %               foo.dot    - file name to view the graphic with the programm
0013 %                           graphviz
0014 %
0015 % VERSION:      $Id: hist2dot.m,v 1.3 2007/07/18 13:58:44 ingo Exp $
0016 %
0017 % HISTORY:      23-05-2007 M Hewitson
0018 %               Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022
0023 %% Write .dot file
0024
0025 fd = fopen(filename, 'w+');
0026
0027 % write header
0028 fprintf(fd, 'digraph G \n{\n');
0029
0030 [n,a, nodes] = getNodes(h, [], 0, 1, []);
0031
0032 % Write block set
0033 fprintf(fd, 'end [label="END"];\n');
0034 for j=1:length(nodes)
0035     node = nodes(j);
0036     % wrap label
0037     % Do some command substitution
0038     fcn = node.names;
0039     shape = 'rectangle';
0040     fsize = 12;
0041     Osize = 24;
0042     opextras = 'style=filled, fillcolor=gray90 fixedsize=true width=0.5';
0043     extras = '';
0044     switch fcn
0045     case 'times'
0046         fcn = '*';
0047         shape = 'circle';
0048         fsize = Osize;
0049         extras = opextras;
0050     case 'mtimes'
0051         fcn = '.*';

```

```

0052     shape = 'circle';
0053     fsize = Osize;
0054     extras = opextras;
0055     case 'plus'
0056         fcn = '+';
0057         shape = 'circle';
0058         fsize = Osize;
0059         extras = opextras;
0060     case 'minus'
0061         fcn = '-';
0062         shape = 'circle';
0063         fsize = Osize;
0064         extras = opextras;
0065     case 'rdivide'
0066         fcn = './';
0067         shape = 'circle';
0068         fsize = Osize;
0069         extras = opextras;
0070     case 'mrdivide'
0071         fcn = '/';
0072         shape = 'circle';
0073         fsize = Osize;
0074         extras = opextras;
0075     end
0076
0077     wstr = wrapstring([fcn char(node.params)], 10);
0078     ss = '';
0079     for s=wstr
0080         ss = [ss '\n' char(s)];
0081     end
0082     ss = ss(3:end);
0083     fprintf(fd, '%s_%d [%s fontsize=%d shape=%s label="%s"];\n', node.names, j,
extras, fsize, shape, ss);
0084 end
0085
0086 % Write node list
0087 fprintf(fd, '\n');
0088 fprintf(fd, '\n');
0089
0090 for j=length(nodes):-1:1
0091 %     disp('-----')
0092     node = nodes(j);
0093     if node.pn > 0
0094         dst = nodes(node.pn);
0095         fprintf(fd, '%s_%d -> %s_%d;\n', node.names, j, dst.names, dst.n);
0096     else
0097         fprintf(fd, '%s_%d -> end;\n', node.names, j);
0098     end
0099 %     disp('-----')
0100 end
0101
0102

```



```
0103 % close
0104 fprintf(fd, '{}\n');
0105
0106 % Close
0107 fclose(fd);
0108
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [noisegenerator](#) > nginit.m

nginit

PURPOSE

SYNOPSIS

```
function y = nginit(Tinit)
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [ltlda_noisegen](#) LTPDA_NOISEGEN uses pzm2ab and franklin to generate a time-series

SOURCE CODE

```
0001 function y = nginit(Tinit)
0002 n = length(Tinit);
0003 %% writing the generator
0004 r = randn(n,1);
0005 y = Tinit * r;
```

Generated on Tue 22-Jan-2008 10:39:13 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 1.2 2008/01/15 17:36:36 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for timeformat objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for timeformat objects.
0007 %
```

```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:       $Id: subsref.m,v 1.2 2008/01/15 17:36:36 ingo Exp $
0011 %
0012 % HISTORY:       31-01-07 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsref.m,v 1.2 2008/01/15 17:36:36 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'timeformat') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             B = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             B = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 B = builtin('subsref', A, S);

```

[Home](#) > [classes](#) > [@ao](#) > [private](#) > aooperate.m

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 1.17 2008/01/10 21:25:13 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function [a1,a2,do] = aooperate(varargin)
0002
0003 % AOOPERATE implements specified operator overload for AOs.
0004 %
0005 % M Hewitson 16-03-07
0006 %
0007 % $Id: aooperate.m,v 1.17 2008/01/10 21:25:13 hewitson Exp $
0008 %
0009
0010 ALGONAME = mfilename;
0011 VERSION  = '$Id: aooperate.m,v 1.17 2008/01/10 21:25:13 hewitson Exp $';
0012
0013 % should be possible to operate with
0014 %      cdata and tsdata - only need to be same length if length(cdata)>1
0015 % or   cdata and fsdata - only need to be same length if length(cdata)>1
0016 % or   cdata and cdata
0017 %
0018 % ** how do we deal with units for these different types?
0019 %   - only compare units when types are the same
```

```

0020
0021 op      = varargin{end};
0022 varargin = varargin{1:end-1};
0023 nargin   = length(varargin);
0024
0025 as = [];
0026 for j=1:nargin
0027     a = varargin{j};
0028     if isa(a, 'ao')
0029
0030         if any(size(a) > 1)
0031             error('### The operation works only with single analysis objects. Do not
use vector or matrices of analysis objects.')
0032         end
0033
0034         as = [as a];
0035     else
0036         if isnumeric(a)
0037             % create an ao from the values
0038             as = [as ao(a)];
0039         else
0040             error('### AO/plus inputs must be either numeric or AO.');

```

```

0069 %%%%%%%%%% get data %%%%%%%%%%%%%%
0070 [x1, y1] = get_xy_values(a1.data);
0071 [x2, y2] = get_xy_values(a2.data);
0072 ny1 = length(y1);
0073 ny2 = length(y2);
0074
0075 % Some special cases
0076 % switch op
0077 %     case '*'
0078 %         if size(y1) == size(y2)
0079 %             y1 = y1.';
0080 %         end
0081 % end
0082
0083
0084 %%%%%%%%%%%%%% check data lengths %%%%%%%%%%%%%%
0085 if isa(d1, 'cdata') && ~isa(d2, 'cdata')
0086     if ny1 > 1 && ny1~=ny2
0087         error('###1 cdata type must be same length as other A0, or single valued.');
```

```

0088     end
0089 elseif isa(d2, 'cdata') && ~isa(d1, 'cdata')
0090     if ny2 > 1 && ny1~=ny2
0091         error('###2 cdata type must be same length as other A0, or single valued.');
```

```

0092     end
0093 end
0094
0095 if ny1~=ny2
0096     if ny1~=1 && ny2~=1
0097         error('### data vectors must be the same length or one should be single
valued.');
```

```

0098     end
0099 end
0100
0101 %%%%%%%%%%%%%% check units %%%%%%%%%%%%%%
0102 if isa(d1, 'fsdata') && isa(d2, 'fsdata')
0103     if ~strcmp(d1.xunits, d2.xunits)
0104         error('### xunits must be the same for same type of A0s.');
```

```

0105     end
0106     % check X-axis
0107     if d1.x ~= d2.x
0108         error('### frequency vectors are not the same.');
```

```

0109     end
0110 end
0111 if isa(d1, 'tsdata') && isa(d2, 'tsdata')
0112     if ~strcmp(d1.xunits, d2.xunits)
0113         error('### xunits must be the same for same type of A0s.');
```

```

0114     end
0115 end
0116 if isa(d1, 'xydata') && isa(d2, 'xydata')
0117     if ~strcmp(d1.xunits, d2.xunits)
0118         error('### xunits must be the same for same type of A0s.');
```

```

0119     end
0120 end

```

```

0121
0122 %%%%%%%%%% Add the data %%%%%%%%%%
0123 % disp(sprintf('*** applying %s', op))
0124 eval(sprintf('y = y1%sy2;', op));
0125
0126 %%%%%%%%%% Create output data object %%%%%%%%%%
0127
0128 if isa(d1, 'fsdata') || isa(d2, 'fsdata')
0129     % then we create an output fsdata
0130     % select which object gives properties to
0131     % the output object
0132     if isa(d1, 'fsdata')
0133         d = d1;
0134     else
0135         d = d2;
0136     end
0137     % then we create an output tsdata
0138     do = fsdata(d.x, y, d.fs);
0139     do = set(do, 'name', sprintf('(%s) %s (%s)', d1.name, op, d2.name));
0140     do = set(do, 'xunits', d.xunits);
0141     do = set(do, 'yunits', d.yunits);
0142     do = set(do, 'enbw', d.enbw);
0143
0144 elseif isa(d1, 'tsdata') || isa(d2, 'tsdata')
0145     % select which object gives properties to
0146     % the output object
0147     if isa(d1, 'tsdata')
0148         d = d1;
0149     else
0150         d = d2;
0151     end
0152     % then we create an output tsdata
0153     do = d;
0154     do = set(do, 'y', y);
0155     do = set(do, 'name', sprintf('(%s) %s (%s)', d1.name, op, d2.name));
0156
0157 elseif isa(d1, 'xydata') || isa(d2, 'xydata')
0158     % select which object gives properties to
0159     % the output object
0160     if isa(d1, 'xydata')
0161         d = d1;
0162     else
0163         d = d2;
0164     end
0165     % then we create an output xydata
0166     do = xydata(d.x, y);
0167     do = set(do, 'name', sprintf('(%s) %s (%s)', d1.name, op, d2.name));
0168     do = set(do, 'xunits', d.xunits);
0169     do = set(do, 'yunits', d.yunits);
0170
0171 elseif isa(d1, 'cdata') || isa(d2, 'cdata')
0172     % then we create an output cdata
0173     % select which object gives properties to

```



```
0174 % the output object
0175 if isa(d1, 'cdata')
0176     d = d1;
0177 else
0178     d = d2;
0179 end
0180 % then we create an output cdata
0181 do = cdata(y);
0182 do = set(do, 'name', sprintf('(%s) %s (%s)', d1.name, op, d2.name));
0183 do = set(do, 'xunits', d.xunits);
0184 do = set(do, 'yunits', d.yunits);
0185
0186 else
0187     error('### unknown data type - 2');
0188 end
0189
0190
0191 % END
```

[Home](#) > [classes](#) > [@history](#) > hist2m.m

hist2m

PURPOSE

HIST2M writes a new m-file that reproduces the analysis described in the history object.

SYNOPSIS

function cmds = hist2m(h)

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 1.14 2007/10/22 12:00:39 ingo Exp \$

HISTORY: 06-02-2007 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.
- [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

```

0001 function cmds = hist2m(h)
0002 % HIST2M writes a new m-file that reproduces the analysis described in the
0003 % history object.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: HIST2M writes a new m-file that reproduces the analysis described
0007 %               in the history object.
0008 %
0009 % CALL:        cmds = hist2m(h);
0010 %
0011 % INPUT:       h      - history object
0012 %
0013 % OUTPUT:      cmds - cell array with the commands to reproduce the data
0014 %
0015 % VERSION:     $Id: hist2m.m,v 1.14 2007/10/22 12:00:39 ingo Exp $
0016 %
0017 % HISTORY:     06-02-2007 M Hewitson
0018 %               Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 ALGONAME = mfilename;
0023 VERSION  = '$Id: hist2m.m,v 1.14 2007/10/22 12:00:39 ingo Exp $';
0024
0025 % Created contains the epochtime in milliseconds and CREATED2INT define
0026 % the number of the last digits
0027 CREATED2INT = 1e7;
0028
0029 % get a node list
0030 [n,a, nodes] = getNodes(h, [], 0, 1, []);
0031
0032 % loop over nodes and convert to commands
0033 i = 1;
0034 cmds = {};
0035 while i <= length(nodes)
0036     v = nodes(i).n;
0037     idx = find([nodes(:).pn] == i);
0038     pl = nodes(i).pl;
0039     h  = nodes(i).h;
0040     aoName = mod(get(h.created, 'utc_epoch_milli'),CREATED2INT);
0041     hi      = [nodes(idx).h];
0042     iNames = zeros(size(hi));
0043     for j=1:length(hi)
0044         created = get(hi(j), 'created');

```

```

0045     iNames(j) = mod(get(created, 'utc_epoch_milli'), CREATED2INT);
0046 end
0047 cmd = writeCmd(char(nodes(i).names), pl, aoName, iNames);
0048 % cmd = writeCmd(char(nodes(i).names), pl, aoName, idx);
0049 cmds = [cmds cellstr(cmd)];
0050 i = i + 1;
0051 end
0052
0053 % now find commands that are duplicated after the '=' and remap those
0054 ncmts = length(cmds);
0055 for j = 1:ncmts
0056     cmdj = cmds{j};
0057     % now inspect all other commands prior to this one
0058     for k = j+1:ncmts
0059         cmdk = cmds{k};
0060         if strcmp(cmdj, cmdk)
0061             cmds{j} = '';
0062         end
0063     end
0064 end
0065
0066 % add the final command to produce a_out
0067 alast = deblank(strtok(cmds{1}, '='));
0068 cmds = [cellstr(sprintf('a_out = %s;', alast)) cmds];
0069
0070 %-----
0071 %
0072 % write a command-line
0073 %
0074 function cmd = writeCmd(name, pl, aon, ains)
0075
0076 ainsStr = '';
0077 % for i=length(ains):-1:1
0078 ni = length(ains);
0079 for i=1:ni
0080     ainsStr = [ainsStr sprintf('a%d, ', ains(i))];
0081 end
0082
0083 name = strrep(name, '\_', '_');
0084
0085 % look at the input parameters
0086 if isa(pl, 'plist')
0087     ps = writePlist(pl);
0088 else
0089     np = 0;
0090     ps = '';
0091 end
0092 if strcmp(ps, 'plist([])')
0093     ps = '';
0094 end
0095 pstr = deblank(sprintf('%s%s', ainsStr, ps));
0096

```

```

0097 if ~isempty(pstr)
0098     if pstr(end) == ','
0099         pstr = pstr(1:end-1);
0100     end
0101 end
0102 cmd = sprintf('a%d = %s(%s);', aon, name, pstr);
0103
0104
0105 %-----
0106 function ps = writePlist(pl)
0107
0108 ps = 'plist(';
0109 np = length(pl.params(:));
0110 for i=1:np
0111     p = pl.params(i);
0112     if ~isempty(p.val)
0113
0114         %%% char %%%
0115         if ischar(p.val)
0116             ps = [ps sprintf('param('%s', '%s') ', p.key, p.val)];
0117
0118         %%% numeric %%%
0119         elseif isnumeric(p.val)
0120             ps = [ps sprintf('param('%s', [, p.key) ltpda_mat2str(p.val) ']) ']];
0121
0122         %%% specwin %%%
0123         elseif isa(p.val, 'specwin')
0124             w = p.val;
0125             wstr = string(w);
0126             ps = [ps sprintf('param('%s', %s) ', p.key, wstr) ];
0127
0128         %%% miir %%%
0129         elseif isa(p.val, 'miir')
0130             f = p.val;
0131             fstr = string(f);
0132             ps = [ps sprintf('param('%s', %s) ', p.key, fstr) ];
0133
0134         %%% pzmodel %%%
0135         elseif isa(p.val, 'pzmodel')
0136             pzm = p.val;
0137             fstr = string(pzm);
0138             ps = [ps sprintf('param('%s', %s) ', p.key, fstr) ];
0139
0140         %%% time %%%
0141         elseif isa(p.val, 'time')
0142             tt = p.val;
0143             fstr = string(tt);
0144             ps = [ps sprintf('param('%s', %s) ', p.key, fstr) ];
0145
0146         %%% provenance %%%
0147         elseif isa(p.val, 'provenance')
0148

```

```
0149     %%% history %%%
0150     elseif isa(p.val, 'history')
0151
0152     else
0153         error(['### unknown parameter type: ' p.key ' ' class(p.val)]);
0154     end
0155 end
0156 end
0157 ps = [ps ')]');
0158
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [noisegenerator](#) > ngprop.m

ngprop

PURPOSE

SYNOPSIS

```
function [x y] = ngprop(Tprop, E, num, y, ns)
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [ltpdata_noisegen](#) LTPDA_NOISEGEN uses pzm2ab and franklin to generate a time-series

SOURCE CODE

```
0001 function [x y] = ngprop(Tprop, E, num, y, ns)
0002
0003 lengT = length(Tprop);
0004 lengb = lengT+1;
0005
0006 num=num';
0007 num = [num zeros(1,(lengb-length(num)-1))];
0008
0009
0010 x = zeros(ns,1);
0011 for i=1:ns
0012     r = randn(lengT,1);
0013     y = E * y + Tprop * r;
0014     x(i) = num*y;
```

0015 [end](#)

0016

0017

Generated on Tue 22-Jan-2008 10:39:13 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 1.1 2007/08/13 10:34:11 ingo Exp \$

HISTORY: 31-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for timespan objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for timespan objects.
0007 %
0008 % EXAMPLES: All possible accesses are possible.
0009 %
```

```
0010 % VERSION:      $Id: subsref.m,v 1.1 2007/08/13 10:34:11 ingo Exp $
0011 %
0012 % HISTORY:       31-01-07 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 B = builtin('subsref', A, S);
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@history](#) > history.m

history

PURPOSE

HISTORY History object class constructor.

SYNOPSIS

function h = history(varargin)

DESCRIPTION

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 1.17 2007/12/05 16:26:22 ingo Exp \$

HISTORY: 30-01-2007 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- **history** HISTORY History object class constructor.
- **xmlparse** XMLPARSE creates a history from a history xml file.

This function is called by:

- **history** HISTORY History object class constructor.

SUBFUNCTIONS

- **function h = init()**

SOURCE CODE

```

0001 function h = history(varargin)
0002 % HISTORY History object class constructor.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: HISTORY History object class constructor.
0007 %               Create a history object.
0008 %
0009 % CONSTRUCTOR: h = history()
0010 %               creates a blank history object
0011 %
0012 %               h = history('name', 'version')
0013 %               creates a history object with this name and version
0014 %
0015 %               h = history('name', 'version', plist)
0016 %               creates a history object with this name,
0017 %               version and parameter list.
0018 %
0019 %               h = history('name', 'version', plist, hi)
0020 %               creates a history object with this name, version,
0021 %               parameter list, and vector of input history objects.
0022 %
0023 %               h = history(hi)
0024 %               creates a new history object from the history object hi
0025 %
0026 % VERSION: $Id: history.m,v 1.17 2007/12/05 16:26:22 ingo Exp $
0027 %
0028 % HISTORY: 30-01-2007 M Hewitson
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032
0033 VERSION = '$Id: history.m,v 1.17 2007/12/05 16:26:22 ingo Exp $';
0034
0035 %%%% Is this a 'Params' call? %%%%
0036 if nargin == 2 && isa(varargin{1}, 'history') && ischar(varargin{2})
0037     if strcmp(varargin{2}, 'Params')

```

```

0038     h = plist();
0039     return
0040 elseif strcmp(varargin{2}, 'Version')
0041     h = VERSION;
0042     return
0043 end
0044 end
0045
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% define history properties %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047
0048 function h = init()
0049     h.name      = 'None';
0050     h.version   = VERSION;
0051     h.plist     = plist();
0052     h.inhists   = [];
0053     h.invars    = {};
0054     h.n         = -1;
0055     h.pn        = -1;
0056     h.created   = time;
0057     h.consver   = VERSION;
0058     h = class(h, 'history');
0059 end
0060
0061 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Create history object %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0062
0063 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% h = history() %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0064 % create default history object
0065 if nargin == 0
0066
0067     h = init();
0068
0069 elseif nargin == 1
0070
0071     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Create from XML fragment %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0073         h = fromxml(varargin{1});
0074     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% From File %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0075     elseif ischar(varargin{1})
0076
0077         filename = varargin{1};
0078         [path, name, ext, vers] = fileparts(filename);
0079         switch ext
0080             case '.mat'
0081                 h = load(filename);
0082                 h = h.a;
0083             case '.xml'
0084                 h = xmlparse(history, filename);
0085             otherwise
0086                 error('### Unknown file type.');
```

```

0090
0091     h = init();
0092
0093     hstruct = varargin{1};
0094     h.name    = hstruct.name;
0095     h.version = hstruct.version;
0096     pl = hstruct.plist;
0097     if isstruct(pl)
0098         pl = plist(pl);
0099     end
0100     h.plist    = pl;
0101     inh = hstruct.inhists;
0102     inhists = [];
0103     for j=1:length(inh)
0104         if isstruct(inh(j))
0105             inhists = [inhists history(inh(j))];
0106         else
0107             inhists = [inhists inh(j)];
0108         end
0109     end
0110     h.inhists = inhists;
0111     h.invars  = hstruct.invars;
0112     h.n       = hstruct.n;
0113     h.pn      = hstruct.pn;
0114     created   = hstruct.created;
0115     if isstruct(created)
0116         created = time(created);
0117     end
0118     h.created    = created;
0119     h.consver    = hstruct.consver;
0120
0121     %%%%%%%%%%% h = history(history) %%%%%%%%%%%
0122     else
0123         h = varargin{1};
0124         h.consver = VERSION;
0125     end
0126
0127     %%%%%%%%%%% h = history(name, version) %%%%%%%%%%%
0128 elseif nargin == 2
0129     %%%%%%%%%%% Database
0130     if isa(varargin{1}, 'database')
0131         h = retrieve(varargin{1}, varargin{2:end});
0132     else
0133         h = init();
0134         h.name    = varargin{1};
0135         h.version = varargin{2};
0136     end
0137
0138     %%%%%%%%%%% h = history(name, version, plist) %%%%%%%%%%%
0139 elseif nargin == 3
0140     if isempty(varargin{3})
0141         varargin{3} = plist();

```

```
0142     end
0143
0144     h          = init();
0145     h.name     = varargin{1};
0146     h.version  = varargin{2};
0147     h.plist    = varargin{3};
0148
0149     %%%%%%%%%% h = history(name, version, plist, history) %%%%%%%%%%
0150 elseif nargin == 4
0151     if isempty(varargin{3})
0152         varargin{3} = plist();
0153     end
0154
0155     h          = init();
0156     h.name     = varargin{1};
0157     h.version  = varargin{2};
0158     h.plist    = varargin{3};
0159     h.inhists  = varargin{4};
0160
0161 else
0162     error('### Unknown number of constructor arguments.');
```

[Home](#) > [m](#) > [noisegenerator](#) > ngsetup.m

ngsetup

PURPOSE

```
ALGONAME = mfilename;
```

SYNOPSIS

```
function [Tinit,Tprop,E] = ngsetup(den,fs)
```

DESCRIPTION

```
ALGONAME = mfilename;
VERSION   = '$Id: ngsetup.m,v 1.1 2007/07/31 15:49:12 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

```
0001 function [Tinit,Tprop,E] = ngsetup(den,fs)
0002
0003 % ALGONAME = mfilename;
0004 % VERSION   = '$Id: ngsetup.m,v 1.1 2007/07/31 15:49:12 anneke Exp $';
0005
0006
0007 den=den';
0008 dt = 1/fs;
0009 length(den);
0010
0011 n = length(den)-1;
0012
0013 %% setting up matrix Aij
```



```

0014
0015 m_a = zeros(n,n);
0016 for i = 1:n
0017     for j = 1:n
0018         if j == i+1
0019             m_a(i,j) = 1;
0020         end
0021         if i == n
0022             m_a(i,j) = -den(j);
0023         end
0024     end
0025 end
0026 %% Matrix exponential E
0027 E = expm(m_a*dt);
0028
0029 %% setting up matrix Bij
0030 B = zeros(n,n);
0031 for i=1:n
0032     if rem(i,2) ~= 0
0033         j0 = (i+1)/2;
0034         s = (-1)^(j0+1);
0035         j = j0;
0036         for k=1:2:(n+1)
0037             B(i,j) = s*den(k);
0038             s = -s;
0039             j = j+1;
0040         end
0041     end
0042     if rem(i,2) == 0
0043         j0 = i/2+1;
0044         s = (-1)^j0;
0045         j = j0;
0046         for k=2:2:(n+1)
0047             B(i,j) = s * den(k);
0048             s = -s;
0049             j = j+1;
0050         end
0051     end
0052 end
0053
0054 %% solve B * m = k
0055 m_k = zeros(n,1);
0056 m_k(n) = 0.5;
0057 m_m = B\m_k;
0058
0059 %% filling covariance matrix Cinit

```

```

0060 Cinit = zeros(n,n);
0061 for i=1:n
0062     for j=1:n
0063         if rem((i+j),2) == 0    % even
0064             Cinit(i,j) = (-1)^((i-j)/2) * m_m((i+j)/2);
0065         else
0066             Cinit(i,j) = 0;
0067         end
0068     end
0069 end
0070
0071
0072 %cholesky decomposition
0073
0074 Tinit = chol(Cinit,'lower'); %lower triangular matrix
0075
0076 %% setting up matrix D
0077 N = n*(n+1)/2;
0078
0079 m_d = zeros(N);
0080 g = zeros(n);
0081 for i=1:n
0082     for j=1:n
0083         if i>=j
0084             g(i,j) = (i*i-i)/2+(j);
0085         else
0086             g(i,j) = (j*j-j)/2+(i);
0087         end
0088     end
0089 end
0090
0091 for i=1:n
0092     for j=i:n
0093         for k=1:n
0094             m_d(g(i,j),g(j,k)) = m_d(g(i,j),g(j,k)) + m_a(i,k);
0095             m_d(g(i,j),g(i,k)) = m_d(g(i,j),g(i,k)) + m_a(j,k);
0096         end
0097     end
0098 end
0099
0100
0101 %% setting up q from D * p = q
0102 m_q = zeros(1,g(n,n));
0103 for i=1:n
0104     for j=i:n
0105         if i==n

```

```

0106             m_q(g(i,j)) = (E(n,n))*(E(n,n))-1;
0107         else
0108             m_q(g(i,j)) = (E(i,n)*E(j,n));
0109         end
0110     end
0111 end
0112
0113 m_p = m_d\m_q';
0114
0115 Cprop = zeros(n);
0116 for i=1:n
0117     for j=1:n
0118         Cprop(i,j) = m_p(g(i,j));
0119     end
0120 end
0121 Tprop = chol(Cprop, 'lower');
0122
0123 %% writing the generator
0124 % r = randn(n,1);
0125 % y = Tinit * r;
0126 % x = zeros(Nfft,1);
0127 % for i=1:Nfft
0128 %     r = randn(n,1);
0129 %     y = E * y + Tprop * r;
0130 %     x(i) = a*y;
0131 % end

```

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 1.13 2007/11/23 12:11:09 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for tsdata objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for tsdata objects.
0007 %
```

```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:        $Id: subsref.m,v 1.13 2007/11/23 12:11:09 ingo Exp $
0011 %
0012 % HISTORY:        31-01-07 M Hewitson
0013 %                  Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsref.m,v 1.13 2007/11/23 12:11:09 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'tsdata') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             B = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             B = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 B = builtin('subsref', A, S);
0033

```

[Home](#) > [classes](#) > [@ao](#) > [testing](#) > horzcat.m

horzcat

PURPOSE [^](#)

HORZCAT overload horizontal concatonation [] for AOs.

SYNOPSIS [^](#)

function b = horzcat(varargin)

DESCRIPTION [^](#)

HORZCAT overload horizontal concatonation [] for AOs.

```
usage: b = [a1 a2];
       b = [a1,a2];
```

M Hewitson 02-05-07

CROSS-REFERENCE INFORMATION [^](#)

This function calls:

This function is called by:

SOURCE CODE [^](#)

```
0001 function b = horzcat(varargin)
0002
0003 % HORZCAT overload horizontal concatonation [] for AOs.
0004 %
0005 % usage: b = [a1 a2];
0006 %         b = [a1,a2];
0007 %
0008 % M Hewitson 02-05-07
```

```
0009 %
0010
0011 ALGONAME = mfilename;
0012 VERSION   = '$Id:$';
0013
0014
0015 % Make outputs
0016 k = 1;
0017 for j=1:nargin
0018     varargin{j}
0019     if isa(varargin{j}, 'ao')
0020         a = varargin{j};
0021         h = history(ALGONAME, VERSION, [], get(a, 'hist'));
0022         % make output analysis object
0023         aout = ao(a.data, h);
0024         % set name
0025         aout = set(aout, 'name', get(a, 'name'));
0026         b(k) = aout;
0027         k    = k + 1;
0028     end
0029 end
0030
0031
0032 % END
```

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 1.6 2007/11/23 12:11:09 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for xydata objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for xydata objects.
0007 %
```



```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:        $Id: subsref.m,v 1.6 2007/11/23 12:11:09 ingo Exp $
0011 %
0012 % HISTORY:         31-01-07 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsref.m,v 1.6 2007/11/23 12:11:09 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'xydata') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             B = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             B = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 B = builtin('subsref', A, S);
0033

```

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 1.1 2007/12/24 19:47:06 hewitson Exp \$

HISTORY: 24-12-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for xyzdata objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for xyzdata objects.
0007 %
0008 % EXAMPLES: All possible accesses are possible.
0009 %
```

```

0010 % VERSION:      $Id: subsref.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $
0011 %
0012 % HISTORY:      24-12-07 M Hewitson
0013 %              Creation
0014 %
0015 %%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsref.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'xyzdata') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             B = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             B = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 B = builtin('subsref', A, S);
0033

```

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 1.2 2008/01/03 15:33:49 ingo 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

```
0001 function B = subsref(A, S)
0002 % SUBSREF Define field name indexing for zero objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SUBSREF Define field name indexing for zero objects.
0007 %
```

```

0008 % EXAMPLES:      All possible accesses are possible.
0009 %
0010 % VERSION:       $Id: subsref.m,v 1.2 2008/01/03 15:33:49 ingo Exp $
0011 %
0012 % HISTORY:       31-01-07 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 VERSION = '$Id: subsref.m,v 1.2 2008/01/03 15:33:49 ingo Exp $';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(A, 'zero') && ischar(S)
0021         in = char(S);
0022         if strcmp(in, 'Params')
0023             B = plist;
0024             return
0025         elseif strcmp(in, 'Version')
0026             B = VERSION;
0027             return
0028         end
0029     end
0030 end
0031
0032 B = builtin('subsref', A, S);

```

[Home](#) > [classes](#) > [@mfir](#) > [private](#) > [parseFilterParams.m](#)

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

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function plo = parseFilterParams(pl)
0002
0003 % PARSEFILTERPARAMS parses the input plist and returns a full plist for
0004 % designing a standard IIR filter. Defaults are used for those parameters
0005 % missing from the input plist.
0006 %
0007 % M Hewitson 11-02-09
0008 %
0009
0010 % 'type' - one of 'highpass', 'lowpass', 'bandpass', 'bandreject'.
0011 %         [default: 'lowpass']
0012 % 'gain' - gain of filter [default: 1.0]
0013 % 'fs'   - sample frequency to design for [default: 1Hz]
0014 % 'order' - order of filter [default: 1]
0015 % 'fc'   - corner frequencies. This is a two element vector for bandpass
0016 %         and bandreject filters. [default: 0.25 or [0.1 0.3] ]
0017 % 'Win'   - a window object to use in the design. [default: Hamming]
0018 %
```

```

0019 %
0020
0021 plo = plist();
0022
0023 % type
0024 type = find(pl, 'type');
0025 if isempty(type)
0026     type = 'lowpass';
0027     disp(['- using default type 'lowpass']);
0028 end
0029 plo = append(plo, param('type', type));
0030
0031 % gain
0032 gain = find(pl, 'gain');
0033 if isempty(gain)
0034     gain = 1.0;
0035     disp(['- using default gain ' num2str(gain)]);
0036 end
0037 plo = append(plo, param('gain', gain));
0038
0039 % order
0040 order = find(pl, 'order');
0041 if isempty(order)
0042     order = 64;
0043     disp(['- using default order ' num2str(order)]);
0044 end
0045 if mod(order,2) == 1
0046     warning('!!! reseting filter order to even number (+1)')
0047     order = order + 1;
0048 end
0049 plo = append(plo, param('order', order));
0050
0051 % fc
0052 fc = find(pl, 'fc');
0053 if isempty(fc)
0054     if strcmp(type, 'bandreject') | strcmp(type, 'bandpass')
0055         fc = [0.1 0.2];
0056     else
0057         fc = 0.25;
0058     end
0059     disp(['- using default fc ' num2str(fc)]);
0060 end
0061 plo = append(plo, param('fc', fc));
0062
0063 % fs
0064 fs = find(pl, 'fs');
0065 if isempty(fs)
0066     fs = 10*max(fc);
0067     warning([sprintf('!!! no sample rate specified. Designing for fs=%2.2fHz.',
fs)...
0068             sprintf('\nThe filter will be redesigned later when used.')]');
0069 end

```

```
0070 plo = append(plo, param('fs', fs));
0071
0072 % win
0073 win = find(pl, 'Win');
0074 if isempty(win)
0075     % then we use the default window
0076     win = specwin('Hamming', order+1);
0077 end
0078 if length(win.win) ~= order + 1
0079     warning('!!! setting window length to filter order !!!');
0080     switch win.name
0081         case {'Kaiser', 'Flattop'}
0082             win = specwin(win.name, order + 1, win.psl1);
0083         otherwise
0084             win = specwin(win.name, order + 1);
0085     end
0086 end
0087 plo = append(plo, param('Win', win));
0088
0089
0090
```


[Home](#) > [classes](#) > [@miir](#) > [private](#) > [parseFilterParams.m](#)

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

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function plo = parseFilterParams(pl)
0002
0003 % PARSEFILTERPARAMS parses the input plist and returns a full plist for
0004 % designing a standard IIR filter. Defaults are used for those parameters
0005 % missing from the input plist.
0006 %
0007 % M Hewitson 11-02-09
0008 %
0009
0010 % 'type' - one of 'highpass', 'lowpass', 'bandpass', 'bandreject'.
0011 %         [default: 'lowpass']
0012 % 'gain' - gain of filter [default: 1.0]
0013 % 'fs'   - sample frequency to design for [default: 1Hz]
0014 % 'order' - order of filter [default: 1]
0015 % 'fc'   - corner frequencies. This is a two element vector for bandpass
0016 %         and bandreject filters. [default: 0.25 or [0.1 0.3] ]
0017
0018 plo = plist();
```

```

0019
0020 % type
0021 type = find(pl, 'type');
0022 if isempty(type)
0023     type = 'lowpass';
0024     disp(['- using default type 'lowpass']);
0025 end
0026 plo = append(plo, param('type', type));
0027
0028 % gain
0029 gain = find(pl, 'gain');
0030 if isempty(gain)
0031     gain = 1.0;
0032     disp(['- using default gain ' num2str(gain)]);
0033 end
0034 plo = append(plo, param('gain', gain));
0035
0036 % order
0037 order = find(pl, 'order');
0038 if isempty(order)
0039     order = 1.0;
0040     disp(['- using default order ' num2str(order)]);
0041 end
0042 plo = append(plo, param('order', order));
0043
0044 % fc
0045 fc = find(pl, 'fc');
0046 if isempty(fc)
0047     if strcmp(type, 'bandreject') | strcmp(type, 'bandpass')
0048         fc = [0.1 0.2];
0049     else
0050         fc = 0.25;
0051     end
0052     disp(['- using default fc ' num2str(fc)]);
0053 end
0054 plo = append(plo, param('fc', fc));
0055
0056 % fs
0057 fs = find(pl, 'fs');
0058 if isempty(fs)
0059     fs = 10*max(fc);
0060     warning([sprintf('!!! no sample rate specified. Designing for fs=%2.2fHz.',
fs)...
0061             sprintf('\nThe filter will be redesigned later when used.')]));
0062 end
0063 plo = append(plo, param('fs', fs));
0064
0065 % ripple
0066 ripple = find(pl, 'ripple');
0067 if isempty(ripple)
0068     ripple = 0.5;
0069     disp(['- using default ripple ' num2str(ripple)]);

```

```
0070 end
0071 plo = append(plo, param('ripple', ripple));
0072
0073
0074
0075
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > buildConditions.m

buildConditions

PURPOSE

SYNOPSIS

```
function buildConditions(varargin)
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

- [buildquery](#)
- [getFields](#) get the selected table
- [getTables](#)

This function is called by:

- [cb_condPlusBtn](#)
- [cb_condSubtractBtn](#)
- [cb_connectBtn](#)

SOURCE CODE

```
0001 function buildConditions(varargin)
0002
0003 fontsize = 10;
0004
0005 panel    = varargin{1};
0006 Ppos     = get(panel, 'Position');
0007 mainfig  = findobj('Tag', 'LTPDARepomainfig');
0008 Nc       = getappdata(mainfig, 'Nconditions');
0009 conn     = getappdata(mainfig, 'connection');
0010 Gproperties = getappdata(mainfig, 'Gproperties');
0011
0012 % before deleting, record the table selection
0013 tableSelected = getappdata(mainfig, 'tableSelected')
0014
```

```

0015 delete(get(panel, 'Children'))
0016 drawnow
0017 %-----
0018 %-- Draw Table selection
0019
0020 % get tables
0021 tables = getTables(conn);
0022 Sleft = 10;
0023 Sheight = 20;
0024 Sbottom = Ppos(2) + Ppos(4) - 20 - Sheight;
0025 Swidth = 50;
0026
0027 sth = uicontrol(panel, 'Style', 'text', ...
0028                 'String', 'From', ...
0029                 'FontSize', fontsize, ...
0030                 'BackgroundColor', Gproperties.Gcol, ...
0031                 'HorizontalAlignment', 'left', ...
0032                 'Position', [Sleft Sbottom Swidth Sheight]);
0033
0034 Sleft = Sleft + Swidth + 10;
0035 Swidth = 150;
0036 pmh = uicontrol(panel, 'Style', 'popupmenu', ...
0037                 'String', tables, ...
0038                 'FontSize', fontsize, ...
0039                 'BackgroundColor', Gproperties.Gcol, ...
0040                 'Value', tableSelected, ...
0041                 'Position', [Sleft Sbottom Swidth Sheight], ...
0042                 'Tag', 'tableSelect');
0043
0044 set(pmh, 'Callback', {'cb_tableSelect', pmh, mainfig});
0045
0046 %-----
0047 %-- Draw Fields Selection list
0048
0049 % get fields
0050 fields = getFields(conn);
0051 pos = get(pmh, 'Position');
0052 Sleft = 10;
0053 Sheight = 20;
0054 Sbottom = pos(2) + pos(4) - 40 - Sheight;
0055 Swidth = 50;
0056
0057 sth = uicontrol(panel, 'Style', 'text', ...
0058                 'String', 'Select', ...
0059                 'FontSize', fontsize, ...
0060                 'BackgroundColor', Gproperties.Gcol, ...
0061                 'HorizontalAlignment', 'left', ...
0062                 'Position', [Sleft Sbottom Swidth Sheight]);
0063

```

```

0064 Sleft    = Sleft + Swidth + 10;
0065 Swidth    = 150;
0066 Sheight   = 200;
0067 Sbottom   = pos(2) + pos(4) - 40 - Sheight;
0068 lbh = uicontrol(panel,'Style','listbox',...
0069                 'String', fields,...
0070                 'Value',1,...
0071                 'BackgroundColor', 'w',...
0072                 'FontSize', fontsize,...
0073                 'Max', 1000,...
0074                 'Position',[Sleft Sbottom Swidth Sheight],...
0075                 'Tag', 'fieldsList');
0076
0077 set(lbh, 'Callback', {'cb_fieldSelect', pmh, mainfig});
0078
0079 %-----
0080 %-- Query display
0081 listpos = get(lbh, 'Position');
0082 Sleft    = 10;
0083 Sheight  = 40;
0084 Sbottom  = listpos(2) - 40 - Sheight;
0085 Swidth   = 60;
0086
0087 sth = uicontrol(panel,'Style','text',...
0088                 'String', 'Query:',...
0089                 'FontSize', fontsize,...
0090                 'Max', 5,...
0091                 'BackgroundColor', Gproperties.Gcol,...
0092                 'HorizontalAlignment', 'left',...
0093                 'Position', [Sleft Sbottom Swidth Sheight]);
0094
0095 Sleft    = 10 + Swidth + 20;
0096 Swidth   = 650;
0097 Sheight  = 100;
0098 Sbottom  = listpos(2) - 40 - Sheight;
0099 sth = uicontrol(panel,'Style','edit',...
0100                 'String', '',...
0101                 'FontSize', fontsize,...
0102                 'Max', 5,...
0103                 'BackgroundColor', 'w',...
0104                 'HorizontalAlignment', 'left',...
0105                 'Position', [Sleft Sbottom Swidth Sheight],...
0106                 'Tag', 'queryDisplayTxt');
0107
0108
0109
0110 %-----
0111 %-- Draw conditions
0112
0113 conditions = {'=', '<>', '<=', '>=', '<', '>', 'like', 'not like', 'is'};

```

```

0114 compounds = {'AND', 'OR', 'XOR'};
0115
0116 pos = get(lbh, 'Position');
0117 Sleft = pos(1) + pos(3) + 20;
0118 Sheight = 20;
0119 Sbottom = Ppos(2) + Ppos(4) - 20 - Sheight;
0120 Swidth = 50;
0121
0122 sth = uicontrol(panel, 'Style', 'text', ...
0123                 'String', 'Where', ...
0124                 'FontSize', fontsize, ...
0125                 'BackgroundColor', Gproperties.Gcol, ...
0126                 'HorizontalAlignment', 'left', ...
0127                 'Position', [Sleft Sbottom Swidth Sheight]);
0128
0129
0130 % pos = get(sth, 'Position');
0131 Cheight = 20;
0132
0133 for k=1:Nc
0134     Cleft = pos(1) + pos(3) + 20;
0135     Cbottom = pos(2) + pos(4) - (k-1)*(Cheight+20);
0136     Cwidth = 150;
0137     % Field Dropdown
0138     pmh = uicontrol(panel, 'Style', 'popupmenu', ...
0139                     'String', fields, ...
0140                     'FontSize', fontsize, ...
0141                     'BackgroundColor', Gproperties.Gcol, ...
0142                     'Value', 1, 'Position', [Cleft Cbottom Cwidth Cheight], ...
0143                     'Tag', sprintf('c%02dField', k));
0144     set(pmh, 'Callback', 'buildquery');
0145
0146     % Condition Dropdown
0147     Cleft = Cleft + Cwidth + 20;
0148     Cwidth = 100;
0149     pmh = uicontrol(panel, 'Style', 'popupmenu', ...
0150                     'String', conditions, ...
0151                     'FontSize', fontsize, ...
0152                     'BackgroundColor', Gproperties.Gcol, ...
0153                     'Value', 1, 'Position', [Cleft Cbottom Cwidth Cheight], ...
0154                     'Tag', sprintf('c%02dCondition', k));
0155     set(pmh, 'Callback', 'buildquery');
0156
0157     % Value txt box
0158     Cleft = Cleft + Cwidth + 20;
0159     Cwidth = 150;
0160     sth = uicontrol(panel, 'Style', 'edit', ...
0161                     'String', '', ...
0162                     'FontSize', fontsize, ...
0163                     'BackgroundColor', 'w', ...

```

```

0164     'HorizontalAlignment', 'left', ...
0165     'Position', [Cleft Cbottom Cwidth Cheight], ...
0166     'Tag', sprintf('c%02dValue', k));
0167     set(sth, 'Callback', 'buildquery');
0168
0169     % Add compound statement
0170     if k < Nc
0171
0172         Cleft = Cleft + Cwidth + 20;
0173         Cwidth = 60;
0174         pmh = uicontrol(panel, 'Style', 'popupmenu', ...
0175             'String', compounds, ...
0176             'FontSize', 10, ...
0177             'BackgroundColor', Gproperties.Gcol, ...
0178             'Value', 1, 'Position', [Cleft Cbottom Cwidth Cheight], ...
0179             'Tag', sprintf('c%02dCompound', k));
0180         set(pmh, 'Callback', 'buildquery');
0181
0182     end
0183
0184 end
0185
0186 %-----
0187 %-- add +/- buttons
0188 k = Nc+1;
0189 Bbottom = pos(2) + pos(4) - (k-1)*(Cheight+20);
0190 Bleft = pos(1) + pos(3) + 20;
0191 Bheight = 20;
0192 Bwidth = 20;
0193 pbh = uicontrol(panel, 'Style', 'pushbutton', 'FontSize', fontsize,
0194     'String', '+', ...
0195         'Position', [Bleft Bbottom Bwidth Bheight ], ...
0196         'Tag', 'condPlusBtn');
0197 set(pbh, 'Callback', {'cb_condPlusBtn', pbh, panel, mainfig});
0198 Bleft = pos(1) + pos(3) + 20 + 30;
0199 pbh = uicontrol(panel, 'Style', 'pushbutton', 'FontSize', fontsize,
0200     'String', '-', ...
0201         'Position', [Bleft Bbottom Bwidth Bheight ], ...
0202         'Tag', 'condSubtractBtn');
0203 set(pbh, 'Callback', {'cb_condSubtractBtn', pbh, panel, mainfig});
0204 %-----
0205 %-- order by
0206 k = Nc + 2;
0207 Sleft = pos(1) + pos(3) + 20;
0208 Sheight = 20;
0209 Swidth = 60;
0210 Sbottom = pos(2) + pos(4) - (k-1)*(Cheight+20);
0211 sth = uicontrol(panel, 'Style', 'text', ...

```



```

0212         'String', 'Order by',...
0213         'FontSize', fontsize,...
0214         'BackgroundColor', Gproperties.Gcol,...
0215         'HorizontalAlignment', 'left',...
0216         'Position',[Sleft Sbottom Swidth Sheight]);
0217
0218 % Field Dropdown
0219 Sleft = pos(1) + pos(3) + 20 + Swidth + 20;
0220 Swidth = 100;
0221 pmh = uicontrol(panel,'Style','popupmenu',...
0222     'String', fields,...
0223     'FontSize', fontsize,...
0224     'BackgroundColor', Gproperties.Gcol,...
0225     'Value',1,'Position',[Sleft Sbottom Swidth Sheight],...
0226     'Tag', 'fieldOrderBy');
0227 set(pmh, 'Callback', 'buildquery');
0228
0229 % Desc Dropdown
0230 Sleft = Sleft + 20 + Swidth;
0231 Swidth = 100;
0232 pmh = uicontrol(panel,'Style','popupmenu',...
0233     'String', {'DESC', 'ASC'},...
0234     'FontSize', fontsize,...
0235     'BackgroundColor', Gproperties.Gcol,...
0236     'Value',1,'Position',[Sleft Sbottom Swidth Sheight],...
0237     'Tag', 'sortDir');
0238 set(pmh, 'Callback', 'buildquery');
0239
0240
0241 %-----
0242 %-- Run query
0243
0244 k = Nc + 3;
0245 Sleft = Ppos(1) + 10;
0246 Sheight = 20;
0247 Swidth = 100;
0248 Sbottom = Ppos(2) + 10;
0249 pbh = uicontrol(panel,'Style','pushbutton','String','Execute query',...
0250     'Position',[Sleft Sbottom Swidth Sheight ],...
0251     'Tag', 'refreshListBtn');
0252 set(pbh, 'Callback', {'cb_executeQuery', pbh, mainfig});
0253
0254
0255 % Build query
0256 buildquery()
0257
0258 drawnow
0259

```


[Home](#) > [classes](#) > [@pole](#) > [private](#) > pfq2ri.m

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 1.8 2008/01/17 16:00:06 anneke Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function ri= pfq2ri(f0, Q)
0002
0003 % Convert frequency/Q pole/zero representation into real
0004 % and imaginary conjugate pairs. Returns [ri conj(ri)]
0005 %
```

```
0006 % ri= pfq2ri(f0, Q)
0007 %
0008 % M Hewitson 02-04-07
0009 %
0010 %
0011 % $Id: pfq2ri.m,v 1.8 2008/01/17 16:00:06 anneke Exp $
0012 %
0013
0014 if(nargin==0)
0015     disp('usage: ri = fq2ri2(f0, Q)')
0016     return
0017 elseif(nargin==1 || nargin==2 && isnan(Q))
0018     ri = -(1/(2*pi*f0));
0019 elseif(nargin==2 && Q>=0)
0020     if Q < 0.5
0021         disp('Q < 0.5! For now treated as if pole is real;')
0022         ri = -(1/(2*pi*f0));
0023     else
0024         w0 = 2*pi*f0;
0025         G = 1/(w0*w0);
0026         re = G.*-w0/(2*Q);
0027         im = G.*w0*sqrt(4*Q*Q-1)/(2*Q);
0028         tri = complex(re,im);
0029         ri = [tri conj(tri)];
0030     end
0031 end
0032
0033
0034
0035 % END
```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > buildquery.m

buildquery

PURPOSE

SYNOPSIS

```
function buildquery()
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [buildConditions](#)
- [cb_fieldSelect](#) Build query
- [cb_tableSelect](#)

SOURCE CODE

```
0001 function buildquery()
0002
0003
0004 mainfig      = findobj('Tag', 'LTPDARepomainfig');
0005
0006 % Get number of conditions
0007 Nc  = getappdata(mainfig, 'Nconditions');
0008
0009 % Select fields
0010 fieldList = findobj('Tag', 'fieldsList');
0011 fieldsStr = get(fieldList, 'String');
0012 vals      = get(fieldList, 'Value');
```

```

0013 fields      = fieldsStr(vals);
0014
0015 % Get table
0016 tableSel = findobj('Tag', 'tableSelect');
0017 tbls     = get(tableSel, 'String');
0018 tbl      = tbls{get(tableSel, 'Value')};
0019
0020 % Start the query
0021 q = 'SELECT ';
0022 for j=1:length(fields)
0023     q = [q fields{j} ','];
0024 end
0025 q = [q(1:end-1) ' FROM ' tbl];
0026
0027
0028 for k=1:Nc
0029
0030     if k==1
0031         q = [q ' WHERE '];
0032     end
0033
0034     % get field
0035     h = findobj('Tag', sprintf('c%02dField', k));
0036     fields = get(h, 'String');
0037     field  = fields{get(h, 'Value')};
0038
0039     q = [q field];
0040
0041     % get condition
0042     h = findobj('Tag', sprintf('c%02dCondition', k));
0043     conds = get(h, 'String');
0044     cond  = conds{get(h, 'Value')};
0045
0046     q = [q ' ' cond ' '];
0047
0048     % get value
0049     h = findobj('Tag', sprintf('c%02dValue', k));
0050     val = get(h, 'String');
0051     q = [q '""' val '""'];
0052
0053     % Get compound
0054     if k<Nc
0055

```

```
0056     h = findobj('Tag', sprintf('c%02dCompound', k));
0057     comps = get(h, 'String');
0058     comp = comps{get(h, 'Value')};
0059
0060     q = [q ' ' comp ' '];
0061
0062 end
0063
0064
0065 end
0066
0067 % Order by
0068 h = findobj('Tag', 'fieldOrderBy');
0069 fields = get(h, 'String');
0070 field = fields{get(h, 'Value')};
0071
0072 q = [q ' ORDER BY ' field];
0073
0074 h = findobj('Tag', 'sortDir');
0075 dirs = get(h, 'String');
0076 dir = dirs{get(h, 'Value')};
0077
0078 q = [q ' ' dir ';'];
0079
0080 % Set query string on GUI
0081 h = findobj('Tag', 'queryDisplayTxt');
0082 set(h, 'String', q);
0083
0084 drawnow
```

[Home](#) > [classes](#) > [@zero](#) > [private](#) > pfq2ri.m

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 1.8 2008/01/17 15:59:42 anneke Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function ri= pfq2ri(f0, Q)
0002
0003 % Convert frequency/Q pole/zero representation into real
0004 % and imaginary conjugate pairs. Returns [ri conj(ri)]
0005 %
```



```
0006 % ri= pfq2ri(f0, Q)
0007 %
0008 % M Hewitson 02-04-07
0009 %
0010 %
0011 % $Id: pfq2ri.m,v 1.8 2008/01/17 15:59:42 anneke Exp $
0012 %
0013
0014 if(nargin==0)
0015     disp('usage: ri = fq2ri2(f0, Q)')
0016     return
0017 elseif(nargin==1 || nargin==2 && isnan(Q))
0018     ri = -(1/(2*pi*f0));
0019 elseif(nargin==2 && Q>=0)
0020     if Q < 0.5
0021         disp('Q < 0.5! For now treated as if zero is real;')
0022         ri = -(1/(2*pi*f0));
0023     else
0024         w0 = 2*pi*f0;
0025         G = 1/(w0*w0);
0026         re = G.*-w0/(2*Q);
0027         im = G.*w0*sqrt(4*Q*Q-1)/(2*Q);
0028         tri = complex(re,im);
0029         ri = [tri conj(tri)];
0030     end
0031 end
0032
0033
0034
0035 % END
0036
```

[Home](#) > [classes](#) > [@cdata](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.2 2007/11/14 18:18:36 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```

```

0002 % ISFIELD tests if the given field is one of the object properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0007 %                properties.
0008 %
0009 % CALL:         isfield(a, 'field');
0010 %
0011 %                The following call returns a parameter list object that
0012 %                contains the default parameter values:
0013 %                >> pl = isfield(cdata, 'Params')
0014 %
0015 %                The following call returns the cvs version number:
0016 %                >> ver = isfield(cdata, 'Version')
0017 %
0018 % VERSION:      $Id: isfield.m,v 1.2 2007/11/14 18:18:36 ingo Exp $
0019 %
0020 % HISTORY: 25-09-07 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: isfield.m,v 1.2 2007/11/14 18:18:36 ingo Exp $';
0026
0027 % 'Params' Call
0028 if nargin == 2
0029     if isa(varargin{1}, 'cdata') && strcmp(varargin{2}, 'Params')
0030         varargout{1} = plist();
0031         return
0032     elseif isa(varargin{1}, 'cdata') && strcmp(varargin{2}, 'Version')
0033         varargout{1} = VERSION;
0034         return
0035     end
0036 end
0037
0038 as = [];
0039 field = '';
0040 for j=1:nargin
0041     if isa(varargin{j}, 'cdata')
0042         as = [as varargin{j}];
0043     end
0044     if ischar(varargin{j})
0045         field = varargin{j};
0046     end
0047 end
0048
0049 res = zeros(size(as));
0050 for j=1: numel(as)
0051     try
0052         t = as(j).(field);
0053         res(j) = 1;

```

```
0054     catch
0055         res(j) = 0;
0056     end
0057 end
0058
0059 varargout{1} = res;
0060
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [analyses](#) > [stuttgart](#) > test_dkalman.m

test_dkalman

PURPOSE

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

SYNOPSIS

This is a script file.

DESCRIPTION

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
testfile for discrete kalman filtering
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION

This function calls:

- [dkalman_matrices](#) %%
- [e2e2ao](#) %%
- [ltpda_dkalman](#) LTPDA_DKALMAN

This function is called by:

SOURCE CODE

```
0001 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0002 % testfile for discrete kalman filtering
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004
0005 clear all;
```

```
0006 close all;
0007
0008 e2e2ao
0009 dkalman_matrices
0010 [X1, X2, X3] = ltpda_dkalman(PHI, Bd, H, 0, Q, R, P0, x0, X1, X12, Fsus)
0011
0012 figure;plot(X1)
0013 plot(X3)
0014
0015 psdX3 = ltpda_pwelch(X3, 'Nolap')
0016 figure;plot(psdX3)
0017
0018
0019
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

cat

PURPOSE ^

CAT concatonate params into a vector.

SYNOPSIS ^

function b = cat(varargin)

DESCRIPTION ^

CAT concatonate params into a vector.

%%%

DESCRIPTION: CAT concatonate params into a vector.

CALL: p_vector = cat(p1, p2, p3);

VERSION: \$Id: cat.m,v 1.3 2007/12/11 17:27:09 ingo 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 ^

```
0001 function b = cat(varargin)
0002 % CAT concatonate params into a vector.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CAT concatonate params into a vector.
0007 %
```

```

0008 % CALL:          p_vector = cat(p1, p2, p3);
0009 %
0010 % VERSION:        $Id: cat.m,v 1.3 2007/12/11 17:27:09 ingo Exp $
0011 %
0012 % HISTORY:         27-03-2007 M Hewitson
0013 %                   Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: cat.m,v 1.3 2007/12/11 17:27:09 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(varargin{1}, 'param') && ischar(varargin{2})
0022         in = char(varargin{2});
0023         if strcmp(in, 'Params')
0024             b = plist;
0025             return
0026         elseif strcmp(in, 'Version')
0027             b = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 b = [];
0034 for j=1:nargin
0035     if isa(varargin{j}, 'param')
0036         b = [b varargin{j}];
0037     end
0038 end
0039

```


[Home](#) > [classes](#) > [@fsdata](#) > isfield.m

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 cvs version number:

```
>> ver = isfield(fsdata, 'Version')
```

VERSION: \$Id: isfield.m,v 1.2 2007/11/14 18:18:36 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```

```

0002 % ISFIELD tests if the given field is one of the object properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0007 %                properties.
0008 %
0009 % CALL:         isfield(a, 'field');
0010 %
0011 %                The following call returns a parameter list object that
0012 %                contains the default parameter values:
0013 %                >> pl = isfield(fsdata, 'Params')
0014 %
0015 %                The following call returns the cvs version number:
0016 %                >> ver = isfield(fsdata, 'Version')
0017 %
0018 % VERSION:      $Id: isfield.m,v 1.2 2007/11/14 18:18:36 ingo Exp $
0019 %
0020 % HISTORY: 25-09-07 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: isfield.m,v 1.2 2007/11/14 18:18:36 ingo Exp $';
0026
0027 % 'Params' Call
0028 if nargin == 2
0029     if isa(varargin{1}, 'fsdata') && strcmp(varargin{2}, 'Params')
0030         varargout{1} = plist();
0031         return
0032     elseif isa(varargin{1}, 'fsdata') && strcmp(varargin{2}, 'Version')
0033         varargout{1} = VERSION;
0034         return
0035     end
0036 end
0037
0038 as = [];
0039 field = '';
0040 for j=1:nargin
0041     if isa(varargin{j}, 'fsdata')
0042         as = [as varargin{j}];
0043     end
0044     if ischar(varargin{j})
0045         field = varargin{j};
0046     end
0047 end
0048
0049 res = zeros(size(as));
0050 for j=1: numel(as)
0051     try
0052         t = as(j).(field);
0053         res(j) = 1;

```

```
0054     catch
0055         res(j) = 0;
0056     end
0057 end
0058
0059 varargout{1} = res;
0060
```

[Home](#) > [m](#) > [helper](#) > phase.m

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

```

0001 function p = phase(resp)
0002 % PHASE return the phase in degrees for a given complex input.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: PHASE return the phase in degrees for a given complex
0007 %               input. Supposed to be analogous to angle but return degrees
0008 %               instead of radians.
0009 %
0010 % COMMENT: This is just a wrapper to call ltpda_phase.
0011 %
0012 % CALL:        p = phase(resp)
0013 %
0014 % INPUTS:      resp - complex number
0015 %
0016 % OUTPUTS:     p     - phase
0017 %
0018 % VERSION: $Id: phase.m,v 1.1 2008/01/20 12:48:50 hewitson Exp $
0019 %
0020 % HISTORY: dd-mm-yyyy M Hewitson
0021 %           Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 p = ltpda_phase(resp);
0026
0027 % END

```

time

PURPOSE

TIME time object class constructor.

SYNOPSIS

function varargout = time(varargin)

DESCRIPTION

```
TIME time object class constructor.

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

DESCRIPTION:  TIME time object class constructor.
              Create a time object.

PROPERTIES:   utc_epoch_milli  - Unix epoch time of the utc time.
              timezone         - Timezone of the date/time
              timeformat       - Matlab timeformat of the date/time
              time_str         - Time string of the unix time in the specified
                               timezone.

CONSTRUCTORS: t1 = time()
              t1 = time('t0_default')
              t1 = time('2007-06-06 14:00:05')
              t1 = time(1186135200000)
              t1 = time(plist)
              t1 = time('14:00:00', 'HH:MM:SS')
              t1 = time('T1: 14:00:00', 'T1: HH:MM:SS')

VERSION:      $Id: time.m,v 1.15 2008/01/15 17:36:36 ingo Exp $

HISTORY:      23-07-2007 Diepholz
              Creation

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

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a time object into a string.
- [set](#) SET set time properties.

- **time** TIME time object class constructor.
- **xmlparse** XMLPARSE creates a time from a time xml file.

This function is called by:

- **plus** PLUS overloads - operator for time objects.
- **time** TIME time object class constructor.

SUBFUNCTIONS

- **function tt = init**

SOURCE CODE

```

0001 function varargout = time(varargin)
0002 % TIME time object class constructor.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION:    TIME time object class constructor.
0007 %                Create a time object.
0008 %
0009 % PROPERTIES:    utc_epoch_milli - Unix epoch time of the utc time.
0010 %                timezone        - Timezone of the date/time
0011 %                timeformat      - Matlab timeformat of the date/time
0012 %                time_str        - Time string of the unix time in the specified
0013 %                                timezone.
0014 %
0015 % CONSTRUCTORS:  t1 = time()
0016 %                t1 = time('t0_default')
0017 %                t1 = time('2007-06-06 14:00:05')
0018 %                t1 = time(1186135200000)
0019 %                t1 = time(plist)
0020 %                t1 = time('14:00:00', 'HH:MM:SS')
0021 %                t1 = time('T1: 14:00:00', 'T1: HH:MM:SS')
0022 %
0023 % VERSION:       $Id: time.m,v 1.15 2008/01/15 17:36:36 ingo Exp $
0024 %
0025 % HISTORY:       23-07-2007 Diepholz
0026 %                Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 VERSION = '$Id: time.m,v 1.15 2008/01/15 17:36:36 ingo Exp $';
0031
0032 % Check if this is a call for parameters
0033 if nargin == 2
0034     if isa(varargin{1}, 'time') && ischar(varargin{2})
0035         in = char(varargin{2});
0036         if strcmp(in, 'Params')

```

```

0037     varargout{1} = plist;
0038     return
0039 elseif strcmp(in, 'Version')
0040     varargout{1} = VERSION;
0041     return
0042 end
0043 end
0044 end
0045
0046 import java.util.GregorianCalendar;
0047 import java.util.TimeZone;
0048
0049 %%%%%%%%%%% Get default values %%%%%%%%%%%
0050
0051 setup_timezone = getappdata(0, 'timezone');
0052 setup_format_str = getappdata(0, 'time_format_str');
0053
0054 % REMARK: The default properties must match together.
0055 %         They must represent the same time format.
0056 default_matlab_str = 'yyyy-mm-dd HH:MM:SS.FFF';
0057 default_matlab_nr = -1;
0058 default_java_str = 'yyyy-MM-dd HH:mm:ss.SSS';
0059
0060 %%%%%%%%%%% Get current time %%%%%%%%%%%
0061
0062 % Define local/utc timezone
0063 timezone = TimeZone.getTimeZone(setup_timezone);
0064
0065 % Define a utc calendar
0066 cal_utc = GregorianCalendar(timezone);
0067
0068 % Get the unix time (timezone = UTC) in milliseconds
0069 utc_epoch_milli = cal_utc.getTimeInMillis;
0070
0071 %%%%%%%%%%% define time properties %%%%%%%%%%%
0072
0073 function tt = init
0074     tt.name = 'time-object';
0075     tt.utc_epoch_milli = utc_epoch_milli;
0076
0077     tt.timezone = timezone;
0078
0079     tt.timeformat = timeformat(setup_format_str, ...
0080                               default_matlab_str, ...
0081                               default_matlab_nr, ...
0082                               default_java_str);
0083
0084     t_format = java.text.SimpleDateFormat(default_java_str);
0085     t_format.setTimeZone(tt.timezone);
0086
0087     tt.time_str = char(t_format.format (tt.utc_epoch_milli));
0088     tt.created = tt.time_str;
0089     tt.version = VERSION;

```



```

0090
0091     tt = class(tt, 'time');
0092
0093     % Set the ltpda_startup time format
0094     tt = set(tt, 'timeformat', tt.timeformat.format_str);
0095 end
0096
0097 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Create time object %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0098
0099 %%%%%%%%%%%%%%% t1 = time() %%%%%%%%%%%%%%%
0100 if nargin == 0
0101
0102     tt = init();
0103
0104 elseif nargin == 1
0105
0106     %%%%%%%%%%%%%%% Create from XML fragment %%%%%%%%%%%%%%%
0107     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0108         tt = fromxml(varargin{1});
0109
0110     %%%%%%%%%%%%%%% t1 = time(time-object) %%%%%%%%%%%%%%%
0111     elseif isa(varargin{1}, 'time')
0112
0113         tt = varargin{1};
0114
0115     %%%%%%%%%%%%%%% t1 = time('t0_default') %%%%%%%%%%%%%%%
0116     elseif strcmp(varargin{1}, 't0_default')
0117
0118         tt = init();
0119         tt = set(tt, 'utc_epoch_milli', 0);
0120
0121     %%%%%%%%%%%%%%% t1 = time('2007-08-03 10:00:00') %%%%%%%%%%%%%%%
0122     %%%%%%%%%%%%%%% t1 = time('foo.mat') %%%%%%%%%%%%%%%
0123     %%%%%%%%%%%%%%% t1 = time('foo.xml') %%%%%%%%%%%%%%%
0124     elseif ischar(varargin{1})
0125
0126         % is this a filename?
0127         [path, name, ext, vers] = fileparts(varargin{1});
0128         if ~isempty(ext) && length(ext) == 4 && isempty(str2num(ext(2)))
0129
0130             switch ext
0131                 case '.mat'
0132                     tt = load(varargin{1});
0133                     tt = tt.a;
0134                 case '.xml'
0135                     tt = xmlparse(time(), varargin{1});
0136                 otherwise
0137                     error('### Unknown file type.');

```

```

0142     tt = set(tt, 'time_str', varargin{1});
0143 end
0144
0145 %%%%%%%%%%% t1 = time(1186135200000) %%%%%%%%%%%
0146 elseif isnumeric(varargin{1})
0147
0148     tt = init();
0149     tt = set(tt, 'utc_epoch_milli', varargin{1});
0150
0151 %%%%%%%%%%% t1 = time(struct) %%%%%%%%%%%
0152 elseif isstruct(varargin{1})
0153
0154     tt = init();
0155
0156     fields = fieldnames(varargin{1});
0157     for ii = 1:length(fields)
0158         field = fields{ii};
0159         %%% timeformat -> timeformat-object
0160         if strcmp(field, 'timeformat')
0161             tf = varargin{1}.timeformat;
0162             if isstruct(tf)
0163                 tf = timeformat(tf);
0164             end
0165             tt.timeformat = tf;
0166             %% All other
0167         else
0168             try
0169                 tt.(field) = varargin{1}.(field);
0170             catch
0171                 error('### The field '%s' in the struct is not a time property.',
field)
0172             end
0173         end
0174     end
0175
0176 %%%%%%%%%%% t1 = time(plist) %%%%%%%%%%%
0177 elseif isa(varargin{1}, 'plist')
0178
0179     pl_timezone = find(varargin{1}, 'timezone');
0180     pl_epoch_milli = find(varargin{1}, 'utc_epoch_milli');
0181     pl_time_str = find(varargin{1}, 'time_str');
0182     pl_timeformat = find(varargin{1}, 'timeformat');
0183
0184     tt = init();
0185
0186     if ~isempty(pl_timeformat)
0187         tt = set(tt, 'timeformat', pl_timeformat);
0188     else
0189         warning(['### The parameter list doesn't contain a timeformat.'...
0190             sprintf('\n          ### Use the default timeformat %s'),...
0191             tt.timeformat.setup_format_str]);
0192     end

```

```

0193
0194     if ~isempty(pl_timezone)
0195         tt = set(tt, 'timezone', pl_timezone);
0196     else
0197         warning(['### The parameter list doesn't contain a timezone. '...
0198                 sprintf('\n          ### Use the default timezone ''UTC''')]);
0199         tt = set(tt, 'timezone', 'UTC');
0200     end
0201
0202     if ~isempty(pl_epoch_milli)
0203         tt = set(tt, 'utc_epoch_milli', pl_epoch_milli);
0204     elseif ~isempty(pl_time_str)
0205         tt = set(tt, 'time_str', pl_time_str);
0206
0207         if ~isempty(pl_timeformat)
0208             tt = set(tt, 'timeformat', pl_timeformat);
0209         end
0210
0211     else
0212         error (['### The parameter list must contain '...
0213                ''utc_epoch_milli'' OR ''time_str'''])
0214     end
0215
0216     %%%%%%%%%%% Unknown constructor %%%%%%%%%%%
0217     else
0218         error ('### Unknown constructor');
0219     end
0220
0221 elseif nargin == 2
0222
0223     %%%%%%%%%%% t1 = time('14:00:00', 'HH:MM:SS') %%%%%%%%%%%
0224     if ischar(varargin{1}) && ischar(varargin{2})
0225
0226         tt = init();
0227         tt = set(tt, 'timeformat', varargin{2});
0228         tt = set(tt, 'time_str' , varargin{1});
0229
0230         %%%%%%%%%%% From DATABASE
0231     elseif isa(varargin{1}, 'database')
0232         tt = retrieve(varargin{1}, varargin{2:end});
0233         %%%%%%%%%%% Unknown constructor %%%%%%%%%%%
0234     else
0235         error ('### Unknown constructor');
0236     end
0237
0238 else
0239
0240     error ('### Unknown number of inputs');
0241
0242 end
0243
0244 varargout{1} = tt;

```

```
0245
0246 end % function varargout = time(varargin)
0247
0248
0249
0250
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

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 1.3 2008/01/02 17:58:51 ingo 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

```
0001 function b = cat(varargin)
0002 % CAT concatonate poles into a vector.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CAT concatonate poles into a vector.
0007 %
```

```

0008 % CALL:      pole_vector = cat(pole1, pole2, pole3);
0009 %
0010 % VERSION:    $Id: cat.m,v 1.3 2008/01/02 17:58:51 ingo Exp $
0011 %
0012 % HISTORY:     27-03-2007 M Hewitson
0013 %              Creation
0014 %
0015 %%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: cat.m,v 1.3 2008/01/02 17:58:51 ingo Exp $';
0018
0019 % Check if this is a call for parameters or for the cvs-version number
0020 if nargin == 2
0021     if isa(varargin{1}, 'pole') && ischar(varargin{2})
0022         in = char(varargin{2});
0023         if strcmp(in, 'Params')
0024             b = plist();
0025             return
0026         elseif strcmp(in, 'Version')
0027             b = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 b = [];
0034 for j=1:nargin
0035     if isa(varargin{j}, 'pole')
0036         b = [b varargin{j}];
0037     end
0038 end
0039

```

[Home](#) > [classes](#) > [@history](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.2 2007/12/04 18:16:11 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```

```

0002
0003 % ISFIELD tests if the given field is one of the object properties.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0008 %                properties.
0009 %
0010 % CALL:         isfield(a, 'field');
0011 %
0012 %                The following call returns a parameter list object that
0013 %                contains the default parameter values:
0014 %                >> pl = isfield(history, 'Params')
0015 %
0016 %                The following call returns the cvs version number:
0017 %                >> ver = isfield(history, 'Version')
0018 %
0019 % VERSION:       $Id: isfield.m,v 1.2 2007/12/04 18:16:11 ingo Exp $
0020 %
0021 % HISTORY: 25-09-07 M Hewitson
0022 %                Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 VERSION = '$Id: isfield.m,v 1.2 2007/12/04 18:16:11 ingo Exp $';
0027
0028 % 'Params' Call
0029 if nargin == 2
0030     if isa(varargin{1}, 'history') && strcmp(varargin{2}, 'Params')
0031         varargout{1} = plist();
0032         return
0033     elseif isa(varargin{1}, 'history') && strcmp(varargin{2}, 'Version')
0034         varargout{1} = VERSION;
0035         return
0036     end
0037 end
0038
0039 as = [];
0040 field = '';
0041 for j=1:nargin
0042     if isa(varargin{j}, 'history')
0043         as = [as varargin{j}];
0044     end
0045     if ischar(varargin{j})
0046         field = varargin{j};
0047     end
0048 end
0049
0050 res = zeros(size(as));
0051 for j=1: numel(as)
0052     try
0053         t = as(j).(field);

```



```
0054     res(j) = 1;
0055     catch
0056         res(j) = 0;
0057     end
0058 end
0059
0060 varargout{1} = res;
0061
```

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 1.6 2008/01/15 17:36:36 ingo Exp $

HISTORY:      23-07-2007 Diepholz
              Creation

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

CROSS-REFERENCE INFORMATION

This function calls:

- **char** CHAR convert a timeformat object into a string.
- **timeformat** TIMEFORMAT timeformat object class constructor.
- **xmlparse** XMLPARSE creates a timeformat from a timeformat xml file.

This function is called by:

- **timeformat** TIMEFORMAT timeformat object class constructor.

SUBFUNCTIONS

- **function tf = init()**

SOURCE CODE

```

0001 function varargout = timeformat(varargin)
0002 % TIMEFORMAT timeformat object class constructor.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION:  TIMEFORMAT timeformat object class constructor.
0007 %               Create a timeformat object.
0008 %
0009 % PROPERTIES:   format_str          - current matlab string
0010 %               format_nr          - current matlab number
0011 %               default_matlab_str - default matlab string (read only)
0012 %               default_matlab_nr - default matlab number (read only)
0013 %               default_java_str   - default java string   (read only)
0014 %
0015 % REMARK:       This class is a PRIVATE class and should not be uses by the
0016 %               user.
0017 %               The default properties must match together. They must represent
0018 %               the same time format.
0019 %
0020 % CONSTRUCTORS: timeformat(format_str,
0021 %                           default_matlab_str,
0022 %                           default_matlab_nr,
0023 %                           default_java_str)
0024 %
0025 % VERSION:      $Id: timeformat.m,v 1.6 2008/01/15 17:36:36 ingo Exp $
0026 %
0027 % HISTORY:      23-07-2007 Diepholz
0028 %               Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION = '$Id: timeformat.m,v 1.6 2008/01/15 17:36:36 ingo Exp $';
0033
0034 % Check if this is a call for parameters
0035 if nargin == 2

```

```

0036     if isa(varargin{1}, 'timeformat') && ischar(varargin{2})
0037         in = char(varargin{2});
0038         if strcmp(in, 'Params')
0039             varargout{1} = plist;
0040             return
0041         elseif strcmp(in, 'Version')
0042             varargout{1} = VERSION;
0043             return
0044         end
0045     end
0046 end
0047
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% define time properties %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049
0050 function tf = init()
0051     tf.format_str = '';
0052     tf.format_nr = -1;
0053     tf.default_matlab_str = '';
0054     tf.default_matlab_nr = -1;
0055     tf.default_java_str = '';
0056     tf = class(tf, 'timeformat');
0057 end
0058
0059
0060 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Create timeformat object %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0061
0062 % Create empty time format object
0063 if nargin == 0
0064     tf = init();
0065     varargout{1} = tf;
0066 elseif nargin == 1
0067
0068     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Create from XML fragment %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0069     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0070         tf = fromxml(varargin{1});
0071         varargout{1} = tf;
0072
0073     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% tf = timeformat(timeformat-object) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0074     elseif isa(varargin{1}, 'timeformat')
0075
0076         tf = varargin{1};
0077         varargout{1} = tf;
0078
0079     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% tf = timeformat('foo.mat') %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0080     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% tf = timeformat('foo.xml') %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081     elseif ischar(varargin{1})
0082
0083         filename = varargin{1};
0084         [path, name, ext, vers] = fileparts(filename);
0085         switch ext
0086             case '.mat'
0087                 tf = load(filename);

```

```

0088         tf = tf.a;
0089         case '.xml'
0090             tf = xmlparse(timeformat, filename);
0091         otherwise
0092             error('### Unknown file type.');
```

end

```

0094
0095     varargout{1} = tf;
0096
0097     %%%%%%%%%%% t1 = time(struct) %%%%%%%%%%%
0098     elseif isstruct(varargin{1})
0099
0100         tf = init();
0101
0102         fields = fieldnames(varargin{1});
0103         for ii = 1:length(fields)
0104             field = fields{ii};
0105             try
0106                 tf.(field) = varargin{1}.(field);
0107             catch
0108                 error('### The field '%s' in the struct is not a timeformat
property.', field)
0109             end
0110         end
0111     else
0112         error('### Unknown constructor');
0113     end
0114
0115     varargout{1} = tf;
0116
0117 elseif nargin == 2
0118     %%%%%%%%%%% From DATABASE
0119     if isa(varargin{1}, 'database')
0120         varargout{1} = retrieve(varargin{1}, varargin{2:end});
0121     else
0122         error('### Incorrect constructor call.');
```

end

```

0124
0125 elseif nargin == 4
0126     tf = init();
0127
0128     tf.format_str      = varargin{1};
0129     tf.default_matlab_str = varargin{2};
0130     tf.default_matlab_nr = varargin{3};
0131     tf.default_java_str  = varargin{4};
0132
0133     if strcmp(tf.format_str, tf.default_matlab_str)
0134         tf.format_nr = tf.default_matlab_nr;
0135     end
0136
0137     varargout{1} = tf;
0138 end

```

```
0139
0140
0141
0142 end
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

cat

PURPOSE

CAT concatonate zeros into a vector.

SYNOPSIS

function b = cat(varargin)

DESCRIPTION

CAT concatonate zeros into a vector.

%%%

DESCRIPTION: CAT concatonate zeros into a vector.

CALL: zero_vector = cat(zero1, zero2, zero3)

VERSION: \$Id: cat.m,v 1.3 2008/01/02 17:58:51 ingo Exp \$

HISTORY: 27-03-2007 Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a zero object into a string.

This function is called by:

SOURCE CODE

```
0001 function b = cat(varargin)
0002 % CAT concatonate zeros into a vector.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CAT concatonate zeros into a vector.
0007 %
```

```

0008 % CALL:          zero_vector = cat(zero1, zero2, zero3)
0009 %
0010 % VERSION:       $Id: cat.m,v 1.3 2008/01/02 17:58:51 ingo Exp $
0011 %
0012 % HISTORY:        27-03-2007 Hewitson
0013 %                  Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: cat.m,v 1.3 2008/01/02 17:58:51 ingo Exp $';
0018
0019 % Check if this is a call for parameters or for the cvs-version number
0020 if nargin == 2
0021     if isa(varargin{1}, 'zero') && ischar(varargin{2})
0022         in = char(varargin{2});
0023         if strcmp(in, 'Params')
0024             b = plist();
0025             return
0026         elseif strcmp(in, 'Version')
0027             b = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 b = [];
0034 for j=1:nargin
0035     if isa(varargin{j}, 'zero')
0036         b = [b varargin{j}];
0037     end
0038 end
0039

```


[Home](#) > [classes](#) > [@mfir](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.2 2007/12/06 17:09:15 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```

```

0002
0003 % ISFIELD tests if the given field is one of the object properties.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0008 %                properties.
0009 %
0010 % CALL:         isfield(a, 'field');
0011 %
0012 %                The following call returns a parameter list object that
0013 %                contains the default parameter values:
0014 %                >> pl = isfield(mfir, 'Params')
0015 %
0016 %                The following call returns the cvs version number:
0017 %                >> ver = isfield(mfir, 'Version')
0018 %
0019 % VERSION:       $Id: isfield.m,v 1.2 2007/12/06 17:09:15 ingo Exp $
0020 %
0021 % HISTORY: 25-09-07 M Hewitson
0022 %                Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 VERSION = '$Id: isfield.m,v 1.2 2007/12/06 17:09:15 ingo Exp $';
0027
0028 % 'Params' Call
0029 if nargin == 2
0030     if isa(varargin{1}, 'mfir') && strcmp(varargin{2}, 'Params')
0031         varargout{1} = plist();
0032         return
0033     elseif isa(varargin{1}, 'mfir') && strcmp(varargin{2}, 'Version')
0034         varargout{1} = VERSION;
0035         return
0036     end
0037 end
0038
0039 as = [];
0040 field = '';
0041 for j=1:nargin
0042     if isa(varargin{j}, 'mfir')
0043         as = [as varargin{j}];
0044     end
0045     if ischar(varargin{j})
0046         field = varargin{j};
0047     end
0048 end
0049
0050 res = zeros(size(as));
0051 for j=1: numel(as)
0052     try
0053         t = as(j).(field);

```

```
0054     res(j) = 1;
0055     catch
0056         res(j) = 0;
0057     end
0058 end
0059
0060 varargout{1} = res;
0061
```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_classMethodList.m](#)

cb_classMethodList

PURPOSE

SYNOPSIS

```
function cb_classMethodList(varargin)
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function cb_classMethodList(varargin)
0002
0003
0004 myh      = varargin{1};
0005 mainfig = varargin{4};
0006
0007 id = findobj('Tag', 'methodInfoPanel');
0008
0009 % get selected method
0010 id = findobj('Tag', 'classMethodList');
0011 ostr = get(id, 'String');
0012 oidx = get(id, 'Value');
0013
0014 objStr = ostr(oidx)
0015
0016 id = findobj('Tag', 'methodInfoPanel');
0017
```

```
0018 if length(oidx) == 1
0019     s = help(char(objStr));
0020     set(id, 'String', s);
0021 end
0022
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@miir](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.2 2007/12/06 17:09:15 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```

```

0002
0003 % ISFIELD tests if the given field is one of the object properties.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0008 %                properties.
0009 %
0010 % CALL:         isfield(a, 'field');
0011 %
0012 %                The following call returns a parameter list object that
0013 %                contains the default parameter values:
0014 %                >> pl = isfield(miir, 'Params')
0015 %
0016 %                The following call returns the cvs version number:
0017 %                >> ver = isfield(miir, 'Version')
0018 %
0019 % VERSION:       $Id: isfield.m,v 1.2 2007/12/06 17:09:15 ingo Exp $
0020 %
0021 % HISTORY: 25-09-07 M Hewitson
0022 %                Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 VERSION = '$Id: isfield.m,v 1.2 2007/12/06 17:09:15 ingo Exp $';
0027
0028 % 'Params' Call
0029 if nargin == 2
0030     if isa(varargin{1}, 'miir') && strcmp(varargin{2}, 'Params')
0031         varargout{1} = plist();
0032         return
0033     elseif isa(varargin{1}, 'miir') && strcmp(varargin{2}, 'Version')
0034         varargout{1} = VERSION;
0035         return
0036     end
0037 end
0038
0039 as = [];
0040 field = '';
0041 for j=1:nargin
0042     if isa(varargin{j}, 'miir')
0043         as = [as varargin{j}];
0044     end
0045     if ischar(varargin{j})
0046         field = varargin{j};
0047     end
0048 end
0049
0050 res = zeros(size(as));
0051 for j=1: numel(as)
0052     try
0053         t = as(j).(field);

```

```
0054     res(j) = 1;
0055     catch
0056         res(j) = 0;
0057     end
0058 end
0059
0060 varargout{1} = res;
0061
```


[Home](#) > [classes](#) > [@history](#) > plot.m

plot

PURPOSE

PLOT plots a history object as a tree diagram.

SYNOPSIS

function varargout = plot(varargin)

DESCRIPTION

PLOT plots a history object as a tree diagram.

%%%

DESCRIPTION: PLOT plots a history object as a tree diagram.

CALL: plot (history)
 plot (history, arg)
 plot (axes_handle, ...)

VERSION: \$Id: plot.m,v 1.15 2007/12/05 16:26:22 ingo Exp \$

NOTE: Some of the code below is taken from Mathworks's treeplot.m

HISTORY: 02-02-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a param object into a string.
- [getNodes](#) GETNODES converts a history object to a nodes structure suitable for plotting as a tree.
- [getNodes_plot](#) GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.
- [plot](#) PLOT plots a history object as a tree diagram.
- [set](#) SET set a history property.

This function is called by:

- [plot](#) PLOT plots a history object as a tree diagram.

SUBFUNCTIONS

- function fcname = getFcnName(name)
- function str = getNodeInputs(invars)

SOURCE CODE

```

0001 function varargout = plot(varargin)
0002 % PLOT plots a history object as a tree diagram.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: PLOT plots a history object as a tree diagram.
0007 %
0008 % CALL:          plot (history)
0009 %               plot (history, arg)
0010 %               plot (axes_handle, ...)
0011 %
0012 % VERSION:       $Id: plot.m,v 1.15 2007/12/05 16:26:22 ingo Exp $
0013 %
0014 % NOTE:          Some of the code below is taken from Mathworks's treeplot.m
0015 %
0016 % HISTORY:       02-02-07 M Hewitson
0017 %                 Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: plot.m,v 1.15 2007/12/05 16:26:22 ingo Exp $';
0022
0023 %%%%%%%%%%%%%% Check if this is a call for parameters %%%%%%%%%%%%%%
0024 if nargin == 2
0025     if isa(varargin{1}, 'history') && ischar(varargin{2})
0026         in = char(varargin{2});
0027         if strcmp(in, 'Params')
0028             varargout{1} = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             varargout{1} = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 %% Used call: plot (history, arg)
0038 if isa(varargin{1}, 'history')
0039
0040     hists = varargin{1};
0041     curr_axes = axes;
0042     if nargin >= 2
0043         varargin = varargin(2:end);
0044     else

```

```

0045     varargin = {};
0046 end
0047
0048 %% Used call: plot (curr_axes_handle, history, arg)
0049 elseif ishandle(varargin{1})
0050
0051     curr_axes = varargin{1};
0052     axes(curr_axes);
0053     hists = varargin{2};
0054     if nargin >= 3
0055         varargin = varargin(3:end);
0056     else
0057         varargin = {};
0058     end
0059
0060 else
0061     error(['### the first input should be a history object or an axes handle' ...
0062         sprintf(' varargin{1} = [%s]', varargin{1})]);
0063 end
0064
0065 %% collect the parameter and put them into the parameter list
0066 plot_args = {};
0067 pl         = plist();
0068 args       = varargin;
0069
0070 while ~isempty(args)
0071
0072     if isa(args{1}, 'plist')
0073         arg = args{1};
0074         pl = append(pl, arg);
0075         args = args(2:end);
0076
0077     % Add the key-values to the parameter list
0078     elseif length(args) >= 2
0079         if ischar(args{1})
0080             arg = args{1};
0081             val = args{2};
0082             pl = append(pl, param(arg, val));
0083
0084             plot_args{end+1} = arg;
0085             plot_args{end+1} = val;
0086             args = args(3:end);
0087
0088         else
0089             error('### the key [%s] is not from the type ''char'', char(arg));
0090         end
0091     else
0092         help(mfilename('fullpath'))
0093         error('### There is no key/value pair left. ');
0094     end
0095
0096 end
0097

```

```

0098
0099 %% Comes the call from the browser or from the command window
0100 max_depth = find(pl, 'max_depth');
0101
0102 %% Output handles for figures and axes
0103
0104 hfig = [];
0105 ax    = [];
0106 at    = [];
0107
0108 nhists = length(hists);
0109 for i=1:nhists
0110
0111     hist = hists(i);
0112     disp(sprintf('--- plotting history %s', hist.name));
0113
0114     % convert history object to a node list
0115     if ~isempty(max_depth)
0116         [n,a, nodes] = getNodes_plot(hist, [], 0, 1, [], 0, max_depth);
0117     else
0118         [n,a, nodes] = getNodes(hist, [], 0, 1, []);
0119     end
0120     p = [nodes(:).pn];
0121     [x,y,h]=treelayout(p);
0122     f = find(p~=0);
0123     pp = p(f);
0124     X = [x(f); x(pp); repmat(NaN,size(f))];
0125     Y = [y(f); y(pp); repmat(NaN,size(f))];
0126     X = X(:);
0127     Y = Y(:);
0128
0129     % figure
0130     hfig = [hfig gcf];
0131
0132     % axes objects
0133     a = plot(curr_axes, X, Y, 'r--', x, y, 'ro');
0134     % plot (X,Y) --> die linien
0135     if length(a)>1
0136         set(a(2), 'MarkerFaceColor', 'r');
0137         set(a(2), 'MarkerSize', 8);
0138     end
0139     args = plot_args;
0140     while ~isempty(args)
0141         prop = args{1};
0142         val   = args{2};
0143         args = args(3:end);
0144         for ii = 1:length(a)
0145             set(a(ii), prop, val);
0146         end
0147     end
0148     ax = [ax a];
0149

```

```

0150 % text objects
0151 a = [];
0152 for j=1:length(x)
0153     % node description
0154     fcname = getFcnName(nodes(j).names);
0155     str = ['{\bf\color{blue}' num2str(nodes(j).n) ':{\bf\color{red}' fcname
'\}' char(nodes(j).params)];
0156     nlstr = getNodeInputs(nodes(j).invars);
0157
0158     na = text(x(j), y(j), ...
0159             [wrapstring(strrep(str, '_', '\_'), getappdata(0,
'wrapstringat')) ...
0160             cellstr(['{\color{magenta}' strrep(nlstr, '_', '\_') '}' ])]);
0161     set(na, 'HorizontalAlignment', 'center');
0162     set(na, 'EdgeColor', 'k', 'FontSize', 10);
0163     set(na, 'BackgroundColor', 'w');
0164     set(na, 'Margin', 5);
0165     a = [a na];
0166 end
0167 at = [at a];
0168
0169 % xlabel(['height = ' int2str(h)]);
0170 axis([0 1 0 1]);
0171 box off;
0172 axis off;
0173
0174 end
0175
0176 % Make outputs
0177 if nargout > 0
0178     varargout{1} = hfig;
0179 end
0180 if nargout > 1
0181     varargout{2} = ax;
0182 end
0183 if nargout > 2
0184     varargout{3} = at;
0185 end
0186
0187
0188 %-----
0189 % compute strings to display for function name
0190 function fcname = getFcnName(name)
0191
0192 switch name
0193     case 'mtimes'
0194         fcname = '\times';
0195     case 'times'
0196         fcname = '\times';
0197     case 'plus'
0198         fcname = '+';
0199     case 'minus'

```

```
0200     fcname = '-';
0201     case 'sqrt'
0202         fcname = '\surd';
0203     case 'mrdivide'
0204         fcname = '/';
0205     case 'rdivide'
0206         fcname = '/';
0207     otherwise
0208         fcname = char(name);
0209 end
0210
0211
0212 %-----
0213 % compute strings to display for inputs to nodes
0214 function str = getNodeInputs(invars)
0215
0216 ni = length(invars);
0217 if ni > 0
0218     str = char(invars{1});
0219     for iv=2:ni
0220         s = char(invars{iv});
0221         if ~strcmp(s, 'pl')
0222             str = [str ' ' s];
0223         end
0224     end
0225 else
0226     str = '';
0227 end
0228
0229
```

timespan

PURPOSE

TIMESPAN timespan object class constructor.

SYNOPSIS

function varargout = timespan(varargin)

DESCRIPTION

```
TIMESPAN timespan object class constructor.

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

DESCRIPTION:  TIMESPAN timespan object class constructor.
              Create a timespan object.

PROPERTIES:  start      - Start time of the time span. (type: time class)
              end        - End time of the time span.  (type: time class)
              timeformat - The time format of start- AND end- time.
              timezone   - The timezone of start- AND end- time.

CONSTRUCTORS: t_span = timespan()
               t_span = timespan( time,      time)
               t_span = timespan('time,      '14:00:05')
               t_span = timespan('14:00:00', '14:00:05')
               t_span = timespan('14:00:00', time)
               t_span = timespan('14:00:00', '14:00:05', 'HH:MM:SS')

VERSION:      $Id: timespan.m,v 1.10 2007/10/16 17:37:21 ingo Exp $

HISTORY:      23-07-2007 Diepholz
               Creation

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

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a timespan object into a string.
- [set](#) SET set timespan properties.
- [timespan](#) TIMESPAN timespan object class constructor.

- `xmlparse` XMLPARSE creates a timespan from a timespan xml file.

This function is called by:

- `timespan` TIMESPAN timespan object class constructor.

SUBFUNCTIONS

- `function t_span = init`

SOURCE CODE

```

0001 function varargout = timespan(varargin)
0002 % TIMESPAN timespan object class constructor.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION:  TIMESPAN timespan object class constructor.
0007 %               Create a timespan object.
0008 %
0009 % PROPERTIES:   start      - Start time of the time span. (type: time class)
0010 %               end        - End time of the time span.   (type: time class)
0011 %               timeformat - The time format of start- AND end- time.
0012 %               timezone   - The timezone of start- AND end- time.
0013 %
0014 % CONSTRUCTORS: t_span = timespan()
0015 %               t_span = timespan( time,      time)
0016 %               t_span = timespan('time,      '14:00:05')
0017 %               t_span = timespan('14:00:00', '14:00:05')
0018 %               t_span = timespan('14:00:00', time)
0019 %               t_span = timespan('14:00:00', '14:00:05', 'HH:MM:SS')
0020 %
0021 % VERSION:      $Id: timespan.m,v 1.10 2007/10/16 17:37:21 ingo Exp $
0022 %
0023 % HISTORY:      23-07-2007 Diepholz
0024 %               Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0027
0028 VERSION = '$Id: timespan.m,v 1.10 2007/10/16 17:37:21 ingo Exp $';
0029
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% define time properties %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 function t_span = init
0033     t_span.name      = 'time span-object';
0034     t_span.start     = time();
0035     t_span.end       = time();
0036     t_span.timeformat = t_span.start.timeformat;
0037     t_span.interval  = '';
0038     t_span.timezone  = t_span.start.timezone;
0039     t_span.created   = time();

```



```

0040     t_span.version      = VERSION;
0041
0042     t_span = class (t_span, 'timespan');
0043 end
0044
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Create time object %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046
0047 %%%%%%%%%%%%%%% t_span = timespan() %%%%%%%%%%%%%%%
0048 if nargin == 0
0049
0050     t_span = init;
0051
0052 elseif nargin == 1
0053
0054     %%%%%%%%%%%%%%% Create from XML fragment %%%%%%%%%%%%%%%
0055     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0056         tf = fromxml(varargin{1});
0057         varargout{1} = tf;
0058     elseif ischar(varargin{1})
0059
0060         filename = varargin{1};
0061         [path, name, ext, vers] = fileparts(filename);
0062         switch ext
0063             case '.mat'
0064                 t_span = load(filename);
0065             case '.xml'
0066                 t_span = xmlparse(timespan, filename);
0067             otherwise
0068                 error('### Unknown file type. ');
0069         end
0070
0071         varargout{1} = t_span;
0072
0073     %%%%%%%%%%%%%%% t1 = time(struct) %%%%%%%%%%%%%%%
0074     elseif isstruct(varargin{1})
0075
0076         t_span = init();
0077
0078         fields = fieldnames(varargin{1});
0079         for ii = 1:length(fields)
0080             field = fields{ii};
0081             %%% start -> time-object
0082             if strcmp(field, 'start')
0083                 start_t = varargin{1}.start;
0084                 if isstruct(start_t)
0085                     start_t = time(start_t);
0086                 end
0087                 t_span.start = start_t;
0088             %%% end -> time-object
0089             elseif strcmp(field, 'end')
0090                 end_t = varargin{1}.end;
0091                 if isstruct(end_t)

```

```

0092         end_t = time(end_t);
0093     end
0094     t_span.end = end_t;
0095     %%% timeformat -> timeformat-object
0096     elseif strcmp(field, 'timeformat')
0097         tf = varargin{1}.timeformat;
0098         if isstruct(tf)
0099             tf = timeformat(tf);
0100         end
0101         t_span.timeformat = tf;
0102     %%% All other
0103     else
0104         try
0105             t_span.(field) = varargin{1}.(field);
0106         catch
0107             error('### The field '%s' in the struct is not a timespan
property.', field)
0108         end
0109     end
0110 end
0111
0112     varargout{1} = t_span;
0113
0114 else
0115     error('### Unknown constructor');
0116 end
0117
0118 elseif nargin == 2
0119
0120     %%%%%%%%%%% t_span = timespan('14:00:00', '14:00:05') %%%%%%%%%%%
0121     if ischar(varargin{1}) && ischar(varargin{2})
0122
0123         t_span = init;
0124         t_span.start = time(varargin{1});
0125         t_span.end    = time(varargin{2});
0126
0127         % Adjust the 'timeformat' AND the 'timezone' of the end time to the start
time.
0128         t_span = set(t_span, 'timeformat', t_span.start.timeformat);
0129         t_span = set(t_span, 'timezone',    t_span.start.timezone);
0130
0131         % Compute the interval
0132         t_span = compute_interval(t_span);
0133
0134     %%%%%%%%%%% From DATABASE
0135     elseif isa(varargin{1}, 'database')
0136
0137         t_span = retrieve(varargin{1}, varargin{2:end});
0138
0139     %%%%%%%%%%% t_span = timespan(time, time) %%%%%%%%%%%
0140     elseif isa(varargin{1}, 'time') && isa(varargin{2}, 'time')
0141

```

```

0142     t_span = init;
0143     t_span.start = varargin{1};
0144     t_span.end   = varargin{2};
0145
0146     % Adjust the 'timeformat' AND the 'timezone' of the end time to the start
time.
0147     t_span = set(t_span, 'timeformat', t_span.start.timeformat);
0148     t_span = set(t_span, 'timezone',   t_span.start.timezone);
0149
0150     % Compute the interval
0151     t_span = compute_interval(t_span);
0152
0153     %%%%%%%%%%% t_span = timespan(time, '14:00:05') %%%%%%%%%%%
0154     elseif isa(varargin{1}, 'time') && ischar(varargin{2})
0155
0156         t_span = init;
0157         t_span.start = varargin{1};
0158         t_span.end   = time(varargin{2}, char(t_span.start.timeformat));
0159
0160         % Adjust the 'timeformat' AND the 'timezone' of the end time to the start
time.
0161         t_span = set(t_span, 'timeformat', t_span.start.timeformat);
0162         t_span = set(t_span, 'timezone',   t_span.start.timezone);
0163
0164         % Compute the interval
0165         t_span = compute_interval(t_span);
0166
0167         %%%%%%%%%%% t_span = timespan('14:00:00', time) %%%%%%%%%%%
0168         elseif ischar(varargin{1}) && isa(varargin{2}, 'time')
0169
0170             t_span = init;
0171             t_span.end   = varargin{2};
0172             t_span.start = time(varargin{1}, char(t_span.end.timeformat));
0173
0174             % Adjust the 'timeformat' AND the 'timezone' of the end time to the start
time.
0175             t_span = set(t_span, 'timeformat', t_span.start.timeformat);
0176             t_span = set(t_span, 'timezone',   t_span.start.timezone);
0177
0178             % Compute the interval
0179             t_span = compute_interval(t_span);
0180
0181     else
0182         error(' ### Unknown constructor. ');
0183
0184     end
0185
0186 elseif nargin == 3
0187
0188     %%%%%%%%%%% t_span = timespan('14:00:00', '14:00:05', 'HH:MM:SS') %%%%%%%%%%%
0189     if ischar(varargin{1}) && ischar(varargin{2}) && ischar(varargin{3})
0190

```

```

0191     t_span = init;
0192     t_span.start = time(varargin{1}, varargin{3});
0193     t_span.end    = time(varargin{2}, varargin{3});
0194     t_span.timeformat = t_span.start.timeformat;
0195
0196     % Compute the interval
0197     t_span = compute_interval(t_span);
0198
0199     elseif isa(varargin{1}, 'time') && isa(varargin{2}, 'time') && isa(varargin
{3}, 'timeformat')
0200
0201     t_span = init;
0202     t_span.start      = varargin{1};
0203     t_span.end        = varargin{2};
0204     t_span.timeformat = varargin{3};
0205
0206     t_span.start = set(t_span.start, 'timeformat', t_span.timeformat);
0207     t_span.end   = set(t_span.end,   'timeformat', t_span.timeformat);
0208
0209     % Compute the interval
0210     t_span = compute_interval(t_span);
0211
0212     else
0213         error (' ### Unknown constructor. ');
0214
0215     end
0216
0217 else
0218
0219     error ('### Unknown number of inputs');
0220
0221 end
0222
0223 if t_span.start.utc_epoch_milli > t_span.end.utc_epoch_milli
0224     error('### The start time is larger than the end time. ');
0225 end
0226
0227 varargout{1} = t_span;
0228
0229
0230 end % varargout = timespan(varargin)

```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_condPlusBtn.m](#)

cb_condPlusBtn

PURPOSE

SYNOPSIS

```
function cb_condPlusBtn(varargin)
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

- [buildConditions](#)

This function is called by:

SOURCE CODE

```
0001 function cb_condPlusBtn(varargin)
0002
0003 myh = varargin{1};
0004
0005 mainfig = findobj('Tag', 'LTPDARepomainfig');
0006 Nc = getappdata(mainfig, 'Nconditions');
0007 if Nc >= 4
0008     Nc = 4;
0009     warning('!!! Can''t have more than 5 conditions.');
```

```
0010 end
0011
0012 setappdata(mainfig, 'Nconditions', Nc + 1);
0013 buildConditions(get(myh, 'Parent'));
0014
```

0015

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@param](#) > isfield.m

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')

VERSION: \$Id: isfield.m,v 1.2 2007/12/11 17:27:09 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```

```

0002
0003 % ISFIELD tests if the given field is one of the object properties.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0008 %                properties.
0009 %
0010 % CALL:         isfield(a, 'field');
0011 %
0012 %                The following call returns a parameter list object that
0013 %                contains the default parameter values:
0014 %                >> pl = isfield(param, 'Params')
0015 %
0016 %                The following call returns the cvs version number:
0017 %                >> ver = isfield(param, 'Version')
0018 %
0019 % VERSION:       $Id: isfield.m,v 1.2 2007/12/11 17:27:09 ingo Exp $
0020 %
0021 % HISTORY: 25-09-07 M Hewitson
0022 %                Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 VERSION = '$Id: isfield.m,v 1.2 2007/12/11 17:27:09 ingo Exp $';
0027
0028 % 'Params' Call
0029 if nargin == 2
0030     if isa(varargin{1}, 'param') && strcmp(varargin{2}, 'Params')
0031         varargout{1} = plist();
0032         return
0033     elseif isa(varargin{1}, 'param') && strcmp(varargin{2}, 'Version')
0034         varargout{1} = VERSION;
0035         return
0036     end
0037 end
0038
0039 as = [];
0040 field = '';
0041 for j=1:nargin
0042     if isa(varargin{j}, 'param')
0043         as = [as varargin{j}];
0044     end
0045     if ischar(varargin{j})
0046         field = varargin{j};
0047     end
0048 end
0049
0050 res = zeros(size(as));
0051 for j=1:numel(as)
0052     try
0053         t = as(j).(field);

```



```
0054     res(j) = 1;
0055     catch
0056         res(j) = 0;
0057     end
0058 end
0059
0060 varargout{1} = res;
0061
```

[Home](#) > [classes](#) > [@specwin](#) > plot.m

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 1.5 2008/01/11 14:04:43 ingo 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

```
0001 function varargout = plot(ws, varargin)
0002 % PLOT plots a specwin object.
```

```

0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: PLOT plots a specwin object.
0007 %
0008 % CALL:          plot(specwin)
0009 %              h = plot(specwin)
0010 %
0011 % VERSION:       $Id: plot.m,v 1.5 2008/01/11 14:04:43 ingo Exp $
0012 %
0013 % HISTORY:       06-02-2007 M Hewitson
0014 %                Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: plot.m,v 1.5 2008/01/11 14:04:43 ingo Exp $';
0019
0020 % Check if this is a call for parameters
0021 if nargin == 2
0022     if isa(ws, 'specwin') && ischar(varargin{1})
0023         in = char(varargin{1});
0024         if strcmp(in, 'Params')
0025             varargout{1} = plist();
0026             return
0027         elseif strcmp(in, 'Version')
0028             varargout{1} = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 hl = [];
0035
0036 hold on
0037 grid on;
0038
0039 colors = getappdata(0, 'ltpda_default_plot_colors');
0040
0041 axes_h = gca;
0042 titleStr = '';
0043 legendStr = '';
0044
0045 for i=1:numel(ws)
0046     w = ws(i);
0047     hl = [hl plot(w.win)];
0048     col = colors(mod(i-1, length(colors))+1);
0049     set(hl(end), 'Color', col);
0050     xlabel('sample');
0051     ylabel('amplitude');
0052     titleStr = [titleStr, ltpda_label(w.name), ', '];
0053     lstr = [sprintf('alpha = %g\n', w.alpha)...
0054            sprintf('psll = %g\n', w.psll)...

```

```
0055         sprintf('rov = %g\n', w.rov)...
0056         sprintf('nenbw = %g\n', w.nenbw)...
0057         sprintf('w3db = %g\n', w.w3db)...
0058         sprintf('flatness = %g\n', w.flatness)];
0059     legendStr = [legendStr cellstr(lstr)];
0060 end
0061
0062 legend(legendStr);
0063 titleStr = titleStr(1:end-2);
0064 title(sprintf('Window: %s', titleStr));
0065
0066 if nargout > 0
0067     varargout{1} = hl;
0068 end
0069
0070 hold off
0071
0072 % END
```

tomfir

PURPOSE

TOMFIR approximates a pole/zero model with an FIR filter.

SYNOPSIS

function varargout = tomfir(varargin)

DESCRIPTION

TOMFIR approximates a pole/zero model with an FIR filter.

%%%

DESCRIPTION: TOMFIR approximates a pole/zero model with an FIR filter. The response of the pzmodel is computed using pzmodel/resp with the additional input parameter of param('f1', 0). The final frequency in the response is set automatically from the pzmodel/resp function if not specified as an input. This upper frequency is then taken as the Nyquist frequency and the sample rate of the corresponding fsdata A0 is set accordingly. The function then calls mfir() with the new fsdata A0 as input. The result is an FIR filter designed to have a magnitude response equal to the magnitude response of the pole/zero model. The filter has linear phase and the phase of the pzmodel is ignored.

CALL: f = tomfir(pzm)
f = tomfir(pzm, plist)

PARAMETERS: f2 - the upper frequency. Sets the Nyquist rate used in the conversion. [default taken from pzmodel/resp]
nf - Sets the number of frequency points used in the conversion [default: 1000]

VERSION: \$Id: tomfir.m,v 1.2 2008/01/08 14:57:07 ingo 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

```

0001 function varargout = tomfir(varargin)
0002 % TOMFIR approximates a pole/zero model with an FIR filter.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: TOMFIR approximates a pole/zero model with an FIR filter.
0007 %               The response of the pzmodel is computed using pzmodel/resp with
0008 %               the additional input parameter of param('f1', 0). The final
0009 %               frequency in the response is set automatically from the
0010 %               pzmodel/resp function if not specified as an input. This upper
0011 %               frequency is then taken as the Nyquist frequency and the
0012 %               sample rate of the corresponding fsdata A0 is set accordingly.
0013 %               The function then calls mfir() with the new fsdata A0 as input.
0014 %               The result is an FIR filter designed to have a magnitude response
0015 %               equal to the magnitude response of the pole/zero model. The
0016 %               filter
0017 %               has linear phase and the phase of the pzmodel is ignored.
0018 % CALL:         f = tomfir(pzm)
0019 %               f = tomfir(pzm, plist)
0020 %
0021 % PARAMETERS:   f2    - the upper frequency. Sets the Nyquist rate used in the
0022 %                   conversion. [default taken from pzmodel/resp]
0023 %               nf    - Sets the number of frequency points used in the conversion
0024 %                   [default: 1000]
0025 %
0026 % VERSION:      $Id: tomfir.m,v 1.2 2008/01/08 14:57:07 ingo Exp $
0027 %
0028 % HISTORY:      16-08-07 M Hewitson
0029 %               Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032
0033 VERSION = '$Id: tomfir.m,v 1.2 2008/01/08 14:57:07 ingo Exp $';
0034
0035 % Check if this is a call for parameters
0036 if nargin == 2
0037     if isa(varargin{1}, 'pzmodel') && ischar(varargin{2})
0038         in = char(varargin{2});
0039         if strcmp(in, 'Params')
0040             varargout{1} = plist;
0041             return
0042         elseif strcmp(in, 'Version')

```

```
0043     varargout{1} = VERSION;
0044     return
0045 end
0046 end
0047 end
0048
0049 pl = plist;
0050
0051 if nargin == 1
0052     pzm = varargin{1};
0053 elseif nargin == 2
0054     pzm = varargin{1};
0055     pl = varargin{2};
0056 else
0057     error('### Incorrect inputs. ');
0058 end
0059
0060 % check design parameters
0061 fs = find(pl, 'fs');
0062
0063 % if isempty(nf)
0064 %     nf = 1000;
0065 % end
0066 % % compute response of model
0067 % if isempty(f2)
0068 %     r = resp(pzm, plist('f1', 0, 'nf', nf));
0069 % else
0070 %     r = resp(pzm, plist('f1', 0, 'f2', f2, 'nf', nf));
0071 % end
0072
0073 r = resp(pzm, plist('f1', 0, 'f2', fs/2, 'nf', 1000));
0074
0075 % Set fs
0076 r = set(r, 'fs', fs);
0077
0078 % compute filter
0079 f = mfir(r);
0080
0081 if nargout == 1
0082     varargout{1} = f;
0083 else
0084     error('### Incorrect outputs. ');
0085 end
0086
0087
0088 % END
```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_condSubtractBtn.m](#)

cb_condSubtractBtn

PURPOSE

SYNOPSIS

```
function cb_condSubtractBtn(varargin)
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

- [buildConditions](#)

This function is called by:

SOURCE CODE

```
0001 function cb_condSubtractBtn(varargin)
0002
0003 myh = varargin{1};
0004
0005 mainfig = findobj('Tag', 'LTPDARepomainfig');
0006 Nc = getappdata(mainfig, 'Nconditions');
0007 if Nc < 1
0008     Nc = 1;
0009 end
0010 setappdata(mainfig, 'Nconditions', Nc - 1);
0011
0012 buildConditions(get(myh, 'Parent'));
0013
0014
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@pzmodel](#) > tomiir.m

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 1.5 2008/01/08 14:57:07 ingo 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

```
0001 function f = tomiir(varargin)
0002 % TOMIIR converts a pzmodel to an IIR filter using a bilinear transform.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: TOMIIR converts a pzmodel to an IIR filter using a bilinear
0007 %                transform.
0008 %
0009 % CALL:          f = tomiir(pzm, fs); % construct for this sample frequency fs
0010 %                f = tomiir(pzm, pl); % construct from plist
0011 %
0012 % PARAMETERS:   'fs' - sample frequency
0013 %
0014 % VERSION:      $Id: tomiir.m,v 1.5 2008/01/08 14:57:07 ingo Exp $
0015 %
0016 % HISTORY:      03-04-2007 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: tomiir.m,v 1.5 2008/01/08 14:57:07 ingo Exp $';
0022
0023 % Check if this is a call for parameters
0024 if nargin == 2
0025     if isa(varargin{1}, 'pzmodel') && ischar(varargin{2})
0026         in = char(varargin{2});
0027         if strcmp(in, 'Params')
0028             f = plist;
0029             return
0030         elseif strcmp(in, 'Version')
0031             f = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 pzm = varargin{1};
0038 if ~isa(pzm, 'pzmodel')
0039     error('### first argument should be a pzmodel.');
```

```
0056     end
0057     else
0058         fs = varargin{2};
0059     end
0060 end
0061
0062 % get a and b coefficients
0063 [a,b] = pzm2ab(pzm, fs);
0064
0065 % make MIIR filter
0066 f = miir(a,b,fs);
0067
0068
0069 % END
```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_connectBtn.m](#)

cb_connectBtn

PURPOSE [^](#)

SYNOPSIS [^](#)

```
function cb_connectBtn(varargin)
```

DESCRIPTION [^](#)

CROSS-REFERENCE INFORMATION [^](#)

This function calls:

- [buildConditions](#)

This function is called by:

SOURCE CODE [^](#)

```
0001 function cb_connectBtn(varargin)
0002
0003 myh      = varargin{1};
0004 mainfig = varargin{4};
0005 status  = findobj('Tag', 'statusTxt');
0006 panels  = getappdata(mainfig, 'panels');
0007
0008 % Get conn
0009 conn = getappdata(mainfig, 'connection');
0010
0011 if isempty(conn)
0012
0013     % get hostname
0014     hostname = get(findobj('Tag','serverTxt'), 'String');
0015
0016     % get the database
0017     db = get(findobj('Tag','databaseTxt'), 'String');
0018
0019
0020     % connect
0021     [conn, username] = mysql_connect(hostname, db);
0022     if isa(conn, 'database')
0023         setappdata(mainfig, 'connection', conn);
```

```
0024     setappdata(mainfig, 'username', username);
0025     set(myh, 'String', 'disconnect');
0026     set(status, 'String', sprintf('connected to %s on %s as %s', db, hostname,
username));
0027
0028     % Set tables on query panel
0029     buildConditions(panels(3));
0030
0031     else
0032         set(status, 'String', 'not connected');
0033         error('Invalid connection.');
```

```
0034     end
0035 else
0036     % disconnect
0037     close(conn);
0038     conn = [];
0039     disp('* disconnected.');
```

```
0040     setappdata(mainfig, 'connection', conn);
0041     setappdata(mainfig, 'username', '');
0042     set(myh, 'String', 'connect');
0043     set(status, 'String', 'not connected');
```

```
0044 end
0045
```

[Home](#) > [classes](#) > [@pole](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.2 2008/01/02 17:58:51 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```

```

0002
0003 % ISFIELD tests if the given field is one of the object properties.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0008 %                properties.
0009 %
0010 % CALL:         isfield(a, 'field');
0011 %
0012 %                The following call returns a parameter list object that
0013 %                contains the default parameter values:
0014 %                >> pl = isfield(pole, 'Params')
0015 %
0016 %                The following call returns the cvs version number:
0017 %                >> ver = isfield(pole, 'Version')
0018 %
0019 % VERSION:       $Id: isfield.m,v 1.2 2008/01/02 17:58:51 ingo Exp $
0020 %
0021 % HISTORY: 25-09-07 M Hewitson
0022 %                Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 VERSION = '$Id: isfield.m,v 1.2 2008/01/02 17:58:51 ingo Exp $';
0027
0028 % 'Params' Call
0029 if nargin == 2
0030     if isa(varargin{1}, 'pole') && strcmp(varargin{2}, 'Params')
0031         varargout{1} = plist();
0032         return
0033     elseif isa(varargin{1}, 'pole') && strcmp(varargin{2}, 'Version')
0034         varargout{1} = VERSION;
0035         return
0036     end
0037 end
0038
0039 as = [];
0040 field = '';
0041 for j=1:nargin
0042     if isa(varargin{j}, 'pole')
0043         as = [as varargin{j}];
0044     end
0045     if ischar(varargin{j})
0046         field = varargin{j};
0047     end
0048 end
0049
0050 res = zeros(size(as));
0051 for j=1: numel(as)
0052     try
0053         t = as(j).(field);

```



```
0054     res(j) = 1;
0055     catch
0056         res(j) = 0;
0057     end
0058 end
0059
0060 varargout{1} = res;
0061
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_dbSelect.m](#)

cb_dbSelect

PURPOSE [^](#)

SYNOPSIS [^](#)

```
function cb_dbSelect(varargin)
```

DESCRIPTION [^](#)

CROSS-REFERENCE INFORMATION [^](#)

This function calls:

This function is called by:

SOURCE CODE [^](#)

```
0001 function cb_dbSelect(varargin)
0002
0003
0004 myh          = varargin{1};
0005 mainfig      = findobj('Tag', 'LTPDARepomainfig');
0006
0007 % Get selected DB
0008 dbh = findobj('Tag', 'dbSelect');
0009 dbs = get(dbh, 'String');
0010 db  = dbs{get(dbh, 'Value')};
0011
0012 % set selected DB
0013 dbh = findobj('Tag', 'databaseTxt');
0014 set(dbh, 'String', db);
0015
0016
0017 end
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@provenance](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.2 2008/01/07 10:21:53 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```

```

0002
0003 % ISFIELD tests if the given field is one of the object properties.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0008 %                properties.
0009 %
0010 % CALL:         isfield(a, 'field');
0011 %
0012 %                The following call returns a parameter list object that
0013 %                contains the default parameter values:
0014 %                >> pl = isfield(provenance, 'Params')
0015 %
0016 %                The following call returns the cvs version number:
0017 %                >> ver = isfield(provenance, 'Version')
0018 %
0019 % VERSION:       $Id: isfield.m,v 1.2 2008/01/07 10:21:53 ingo Exp $
0020 %
0021 % HISTORY: 25-09-07 M Hewitson
0022 %                Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 VERSION = '$Id: isfield.m,v 1.2 2008/01/07 10:21:53 ingo Exp $';
0027
0028 % 'Params' Call
0029 if nargin == 2
0030     if isa(varargin{1}, 'provenance') && strcmp(varargin{2}, 'Params')
0031         varargout{1} = plist();
0032         return
0033     elseif isa(varargin{1}, 'provenance') && strcmp(varargin{2}, 'Version')
0034         varargout{1} = VERSION;
0035         return
0036     end
0037 end
0038
0039 as = [];
0040 field = '';
0041 for j=1:nargin
0042     if isa(varargin{j}, 'provenance')
0043         as = [as varargin{j}];
0044     end
0045     if ischar(varargin{j})
0046         field = varargin{j};
0047     end
0048 end
0049
0050 res = zeros(size(as));
0051 for j=1: numel(as)
0052     try
0053         t = as(j).(field);

```

```
0054     res(j) = 1;
0055     catch
0056         res(j) = 0;
0057     end
0058 end
0059
0060 varargout{1} = res;
0061
```

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 1.5 2008/01/14 20:03:53 ingo Exp \$

HISTORY: 03-08-2007 Diepholz
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [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

```

0001 function res = plus(t1,t2)
0002 % PLUS overloads - operator for time objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: PLUS overloads - operator for analysis objects.
0007 %               This function adds a time in seconds to a time object.
0008 %               It is possible to define the addend as a string,
0009 %               number or a time object.
0010 %
0011 % CALL:         t2 = t1 + value (in seconds)
0012 %               sec = t1 + t2;
0013 %
0014 % EXAMPLES:     >> t1      = time('2007-07-01 12:23:33');
0015 %               >> t2      = t1 + 30;
0016 %               >> t3      = t1 + '30';
0017 %               >> seconds = t1 + t2;
0018 %
0019 % VERSION:      $Id: plus.m,v 1.5 2008/01/14 20:03:53 ingo Exp $
0020 %
0021 % HISTORY:      03-08-2007 Diepholz
0022 %               Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 VERSION = '$Id: plus.m,v 1.5 2008/01/14 20:03:53 ingo Exp $';
0027
0028 % 'Params' Call
0029 if nargin == 2
0030     if isa(t1, 'time') && strcmp(t2, 'Params')
0031         res = plist();
0032         return
0033     elseif isa(t1, 'time') && strcmp(t2, 'Version')
0034         res = VERSION;
0035         return
0036     end
0037 end
0038
0039 if isnumeric(t2)
0040     t2 = ltpda_mat2str(t2);
0041 end
0042
0043 %%%%%%%%% >> t2 = t1 + 30; %%%%%%%%%
0044 %%%%%%%%% >> t2 = t1 + '30'; %%%%%%%%%
0045 if ischar(t2)
0046     java_format_str_seconds = 'ss';
0047
0048     utc_timezone = java.util.TimeZone.getTimeZone('UTC');
0049

```



```
0050     t_format_seconds = java.text.SimpleDateFormat(java_format_str_seconds);
0051     t_format_seconds.setTimeZone(utc_timezone);
0052
0053     epoch_time_seconds = t_format_seconds.parse(t2);
0054     epoch_time_seconds = epoch_time_seconds.getTime;
0055
0056     res = set(t1, 'utc_epoch_milli', t1.utc_epoch_milli+epoch_time_seconds);
0057
0058     %%%%    >> number = t1 + t2;    %%%%
0059 elseif isa(t2, 'time')
0060
0061     res = (t1-time(format(t1, 'yyyy-mm-dd')) + (t2-time(format(t2, 'yyyy-mm-
dd'))));
0062
0063 %     res = (t1.utc_epoch_milli + t2.utc_epoch_milli)/1000;
0064
0065 else
0066     error('### Unknown plus-operation to a time object.');
```

tsdata

PURPOSE

TSDATA time-series object class constructor.

SYNOPSIS

function ts = tsdata(varargin)

DESCRIPTION

```
TSDATA time-series object class constructor.

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

DESCRIPTION: TSDATA time-series object class constructor.
             Create a time-series data object.

Properties:
  name      - name of time-series object
  fs        - sample rate of data
  x         - time samples vector
  y         - x samples vector
  nsecs     - the length of this time-series in seconds
  xunits    - units to interpret the time samples (e.g., seconds)
  yunits    - units to interpret the data samples (e.g., Volts)
  t0        - time-stamp of the first data sample in UTC format yyyy-mm-dd HH:MM:SS

             created - creation time of this fsdata object. Created by the function now
             version - version of the constructor code

Possible constructors:
  ts = tsdata()      - creates a blank time-series object
  ts = tsdata(y)     - creates a time-series object with the given
                      x-data.
  ts = tsdata(x,y)   - creates a time-series object with the given
                      (t,x)-data. The sample rate is then set as
                      1/(t(2)-t(1)).
  ts = tsdata(y,fs)  - creates a time-series object with the given
                      x-data. The time vector t[] is grown from the
                      sample rate. The first sample is assigned time 0.

HISTORY: 30-01-2007 Hewitson
          Creation

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

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.
- [tsdata](#) TSDATA time-series object class constructor.
- [xmlparse](#) XMLPARSE creates a tsdata from a tsdata xml file.

This function is called by:

- [tsdata](#) TSDATA time-series object class constructor.

SUBFUNCTIONS

- [function ts = init](#)

SOURCE CODE

```

0001 function ts = tsdata(varargin)
0002 % TSDATA time-series object class constructor.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: TSDATA time-series object class constructor.
0007 %               Create a time-series data object.
0008 %
0009 %   Properties:
0010 %       name      - name of time-series object
0011 %       fs        - sample rate of data
0012 %       x         - time samples vector
0013 %       y         - x samples vector
0014 %       nsecs     - the length of this time-series in seconds
0015 %       xunits    - units to interpret the time samples (e.g., seconds)
0016 %       yunits    - units to interpret the data samples (e.g., Volts)
0017 %       t0        - time-stamp of the first data sample in UTC format yyyy-mm-dd
HH:MM:SS
0018 %       created   - creation time of this fsdata object. Created by the function
now
0019 %       version   - version of the constructor code
0020 %
0021 %   Possible constructors:
0022 %       ts = tsdata()      - creates a blank time-series object
0023 %       ts = tsdata(y)    - creates a time-series object with the given
0024 %                           x-data.
0025 %       ts = tsdata(x,y)  - creates a time-series object with the given
0026 %                           (t,x)-data. The sample rate is then set as
0027 %                           1/(t(2)-t(1)).

```

```

0028 %         ts = tsdata(y,fs) - creates a time-series object with the given
0029 %                               x-data. The time vector t[] is grown from the
0030 %                               sample rate. The first sample is assigned time 0.
0031 %
0032 % HISTORY: 30-01-2007 Hewitson
0033 %         Creation
0034 %
0035 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0036
0037 VERSION = '$Id: tsdata.m,v 1.22 2007/11/30 19:36:05 ingo Exp $';
0038
0039 %%%% Is this a 'Params' call? %%%%
0040 if nargin == 2 && isa(varargin{1}, 'tsdata') && ischar(varargin{2})
0041     if strcmp(varargin{2}, 'Params')
0042         ts = plist();
0043         return
0044     elseif strcmp(varargin{2}, 'Version')
0045         ts = VERSION;
0046         return
0047     end
0048 end
0049
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% define tsdata properties %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051
0052 function ts = init
0053     ts.name      = 'None';
0054     ts.x         = [];
0055     ts.y         = [];
0056     ts.fs        = 0;
0057     ts.xunits    = '';
0058     ts.yunits    = '';
0059     ts.nsecs     = 0;
0060     ts.t0        = time('t0_default');
0061     ts.version   = VERSION;
0062     ts.created   = time;
0063     ts = class(ts, 'tsdata');
0064 end
0065
0066 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Create tsdata object %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0067
0068 %%%%%%%%% ts = tsdata() %%%%%%%%%
0069 % create default tsdata object
0070 if nargin == 0
0071
0072     ts = init();
0073
0074 elseif nargin == 1
0075
0076     %%%%%%%%% Create from XML fragment %%%%%%%%%
0077     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0078         ts = fromxml(varargin{1});
0079     %%%%%%%%% From File %%%%%%%%%

```

```

0080 elseif ischar(varargin{1})
0081
0082     filename = varargin{1};
0083     [path, name, ext, vers] = fileparts(filename);
0084     switch ext
0085     case '.mat'
0086         ts = load(filename);
0087         ts = ts.a;
0088     case '.xml'
0089         ts = xmlparse(tsdata, filename);
0090     otherwise
0091         error('### Unknown file type.');
```

0092 end

0093 %%%%%%%%%%% ts = tsdata(tsdata) %%%%%%%%%%%

```

0094 elseif isa(varargin{1}, 'tsdata')
0095     ts = varargin{1};
0096
0097     %%%%%%%%%%% ts = tsdata(struct) %%%%%%%%%%%
0098 elseif isstruct(varargin{1})
0099
0100     ts = init();
0101
0102     fields = fieldnames(varargin{1});
0103     for ii = 1:length(fields)
0104         field = fields{ii};
0105
0106         %%% created -> time-object
0107         if strcmp(field, 'created')
0108             created = varargin{1}.created;
0109             if isstruct(created)
0110                 created = time(created);
0111             end
0112             ts.created = created;
0113         %%% t0 -> time-object
0114         elseif strcmp(field, 't0')
0115             t0 = varargin{1}.t0;
0116             if isstruct(t0)
0117                 t0 = time(t0);
0118             end
0119             ts.t0 = t0;
0120         %%% All other
0121         else
0122             try
0123                 ts.(field) = varargin{1}.(field);
0124             catch
0125                 error('### The field '%s' in the struct is not a tsdata property.',
0126                     field)
0127             end
0128         end
0129     end
0130     %%%%%%%%%%% ts = tsdata(y_vector) %%%%%%%%%%%
0131 else
```

```

0132
0133     ts = init();
0134
0135     ts.y      = varargin{1};
0136     ts.fs     = 1;
0137     ts.nsecs  = length(ts.y)/ts.fs;
0138
0139     % Unify the y-axis
0140     if size(ts.y,2) > size(ts.y,1)
0141         ts.y = [ts.y].';
0142     end
0143     ts = setTime(ts);
0144
0145 end
0146
0147 elseif nargin == 2
0148
0149     %%%%%%%%% From DATABASE
0150     if isa(varargin{1}, 'database')
0151         ts = retrieve(varargin{1}, varargin{2:end});
0152         return
0153     %%%%%%%%% ts = tsdata(x_vector, y_vector) %%%%%%%%%
0154     elseif length(varargin{1}) == length(varargin{2}) && length(varargin{1})>1
0155
0156         ts = init();
0157
0158         ts.x      = varargin{1};
0159         ts.y      = varargin{2};
0160         ts.fs     = 1.0/(ts.x(min(2,length(ts.x)))-ts.x(1));
0161         ts.nsecs  = ts.x(end) - ts.x(1) + 1/ts.fs;
0162
0163         % Unify the y-axis
0164         if size(ts.y,2) > size(ts.y,1)
0165             ts.y = [ts.y].';
0166         end
0167         ts = reshapeT(ts);
0168
0169     %%%%%%%%% ts = tsdata(y_vector, fs) %%%%%%%%%
0170 else
0171     % check we have fs properly here
0172     if length(varargin{2}) > 1
0173         error(['### unknown constructor call. Either x and y should be the' ...
0174             'same length, or fs should be a single value.']);
0175     end
0176
0177     ts = init();
0178
0179     ts.y      = varargin{1};
0180     ts.fs     = varargin{2};
0181     ts.nsecs  = length(ts.y)/ts.fs;
0182
0183     if size(ts.y,2) > size(ts.y,1)

```

```
0184     ts.y = [ts.y].';
0185     end
0186     ts = setTime(ts);
0187
0188     end
0189 else
0190     error('### Unknown number of constructor arguments. ');
0191 end
0192
0193
0194 end % function ts = tsdata(varargin)
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_executeQuery.m](#)

cb_executeQuery

PURPOSE

Get connection

SYNOPSIS

```
function cb_executeQuery(varargin)
```

DESCRIPTION

Get connection

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function cb_executeQuery(varargin)
0002
0003 % Get connection
0004 mainfig = findobj('Tag', 'LTPDARepomainfig');
0005 conn    = getappdata(mainfig, 'connection');
0006 if isempty(conn)
0007     error('### Please connect to a database first.');
```

0008 end

0009

0010 % Get Selected fields

0011 fieldList = findobj('Tag', 'fieldsList');

0012 fieldsStr = get(fieldList, 'String');

0013 vals = get(fieldList, 'Value');


```
0014 fields      = fieldsStr(vals)';
0015
0016
0017 % Get query
0018 h = findobj('Tag', 'queryDisplayTxt');
0019 q = get(h, 'String');
0020
0021 curs = exec(conn, q);
0022 curs = fetch(curs);
0023 results = curs.Data;
0024 close(curs);
0025
0026 if strcmp(results{1}, 'No Data')
0027     errordlg('Query returned no results.', 'Query Empty');
0028     return
0029 else
0030     whos('results')
0031     whos('fields')
0032     sqlResultsGUI(results, fields, q);
0033 end
0034
```

[Home](#) > [classes](#) > [@pzmodel](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.2 2008/01/07 18:13:10 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```

```

0002 % ISFIELD tests if the given field is one of the object properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0007 %                properties.
0008 %
0009 % CALL:         isfield(a, 'field');
0010 %
0011 %                The following call returns a parameter list object that
0012 %                contains the default parameter values:
0013 %                >> pl = isfield(pzmodel, 'Params')
0014 %
0015 %                The following call returns the cvs version number:
0016 %                >> ver = isfield(pzmodel, 'Version')
0017 %
0018 % VERSION:      $Id: isfield.m,v 1.2 2008/01/07 18:13:10 ingo Exp $
0019 %
0020 % HISTORY: 25-09-07 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: isfield.m,v 1.2 2008/01/07 18:13:10 ingo Exp $';
0026
0027 % 'Params' Call
0028 if nargin == 2
0029     if isa(varargin{1}, 'pzmodel') && strcmp(varargin{2}, 'Params')
0030         varargout{1} = plist();
0031         return
0032     elseif isa(varargin{1}, 'pzmodel') && strcmp(varargin{2}, 'Version')
0033         varargout{1} = VERSION;
0034         return
0035     end
0036 end
0037
0038 as = [];
0039 field = '';
0040 for j=1:nargin
0041     if isa(varargin{j}, 'pzmodel')
0042         as = [as varargin{j}];
0043     end
0044     if ischar(varargin{j})
0045         field = varargin{j};
0046     end
0047 end
0048
0049 res = zeros(size(as));
0050 for j=1: numel(as)
0051     try
0052         t = as(j).(field);
0053         res(j) = 1;

```

```
0054     catch
0055         res(j) = 0;
0056     end
0057 end
0058
0059 varargout{1} = res;
0060
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > uitabdemo.m

uitabdemo

PURPOSE

UITABDEMO This script demonstrates the use of **UITABPANEL**.

SYNOPSIS

function hh = uitabdemo(option)

DESCRIPTION

UITABDEMO This script demonstrates the use of **UITABPANEL**.

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 CreateTab1\(htab,evdt,hpanel,hstatus\)](#)
- [function CreateTab2\(htab,evdt,hpanel,hstatus\)](#)
- [function CreateTab3\(htab,evdt,hpanel,hstatus\)](#)
- [function TabResize1\(hobj,evdt\)](#)
- [function TabResize2\(hobj,evdt,ysiz\)](#)
- [function TabStatus2\(hobj,evdt\)](#)
- [function PlotTheEarth\(topo,topomap1\)](#)
- [function PlotTheGlobe\(hobj,evdt,topo\)](#)

SOURCE CODE

```
0001 function hh = uitabdemo(option)
0002
0003 % UITABDEMO This script demonstrates the use of UITABPANEL.
0004
0005 % Author: Shiyong Zhao (zhao@arch.umsi.edu)
0006 % Version: 1.0
0007 % First created: May 02, 2006
0008 % Last modified: May 20, 2006
```

```

0009
0010 if nargin<1, option = 1; end
0011 switch option
0012     case 1
0013         style1 = 'popup';
0014         style2 = 'lefttop';
0015         style3 = 'rightbottom';
0016     case 2
0017         style1 = 'popup';
0018         style2 = 'righttop';
0019         style3 = 'leftbottom';
0020     otherwise
0021         style1 = 'popup';
0022         style2 = 'centertop';
0023         style3 = 'centerbottom';
0024 end
0025
0026 width = 0;
0027 hfig = figure(...
0028     'Name','uitabpanel demo',...
0029     'NumberTitle','off', ...
0030     'Menubar','none',...
0031     'Toolbar','none');
0032
0033 %-----
0034 % Creating the tabs
0035 %-----
0036 htab(1) = uitabpanel(...
0037     'Parent',hfig,...
0038     'Style',style1,...
0039     'Units','normalized',...
0040     'Position',[0,0,0.3,1],...
0041     'FrameBackgroundColor',[0.4314,0.5882,0.8431],...
0042     'FrameBorderStyle','etchedin',...
0043     'Title',{'This is popup tab','Click me!','See also','About'},...
0044     'PanelHeights',[8,30,10,10],...
0045     'HorizontalAlignment','left',...
0046     'FontWeight','bold',...
0047     'TitleBackgroundColor',[0.9255 0.9490 0.9765],...
0048     'TitleForegroundColor',[0.1294,0.3647,0.8510],...
0049     'PanelBackgroundColor',[0.7725,0.8431,0.9608],...
0050     'PanelBorderStyle','line',...
0051     'CreateFcn',@CreateTab1);
0052
0053 htab(2) = uitabpanel(...
0054     'Parent',hfig,...
0055     'TabPosition',style2,...
0056     'Position',[0.3,0.5,0.7,0.5],...
0057     'Margins',[0,-1,1,0],'pixels',...
0058     'PanelBorderStyle','line',...
0059     'Title',{'Introduction','Syntax','Features','Todo'},...
0060     'CreateFcn',@CreateTab2);

```

```

0061
0062 htab(3) = uitabpanel(...
0063     'Parent',hfig,...
0064     'TitlePosition',style3,...
0065     'Position',[0.3,0,0.7,0.5],...
0066     'Margins',{[0,1,1,0],'pixels'},...
0067     'Title',{'The Earth','The Globe','More Styles'},...
0068     'CreateFcn',@CreateTab3,...
0069     'SelectedItem',3);
0070
0071 if narginout, hh = htab; end
0072
0073 %-----
0074 function CreateTab1(htab,evdt,hpanel,hstatus)
0075     uicontrol(...
0076         'Parent',hpanel(1),...
0077         'Units','normalized',...
0078         'Position',[.1,.1,.9,.8],...
0079         'BackgroundColor',get(hpanel(3),'BackgroundColor'),...
0080         'HorizontalAlignment','left',...
0081         'Style','text',...
0082         'String',[...
0083             'Popup panel is another      '
0084             'type of UITABPANEL          '
0085             'with the ''Style''           '
0086             'property set to ''Popup''.  '
0087             'Click on its title bar      '
0088             'toggle open/close of the   '
0089             'popup panel.                ']);
0090
0091     uicontrol(...
0092         'Parent',hpanel(2),...
0093         'Units','normalized',...
0094         'Position',[.1,.02,.8,.95],...
0095         'BackgroundColor',get(hpanel(3),'BackgroundColor'),...
0096         'HorizontalAlignment','left',...
0097         'Style','text',...
0098         'String',[...
0099             'Here is a long panel.      '
0100             'Using the vertical          '
0101             'slider to scroll up         '
0102             'and down for the rest      '
0103             'of content in this         '
0104             'popup panel.                '
0105             '                            '
0106             'All popup panels have      '
0107             'the same vertical           '
0108             'layout.                    '
0109             '                            '
0110             'The length of each         '
0111             'panel in the group can     '
0112             'be specified using the'

```

```

0113         'the property
0114         '''PanelHeights''' at
0115         'creation time, which
0116         'should be a numeric
0117         'array with the same
0118         'length of '''Title''' in
0119         'the unit of
0120         '''Characters'''.
0121         '
0122         'Interactive change of
0123         'the length of a panel
0124         'in the group can only
0125         'be done when it is
0126         'deactivated.']);
0127
0128     uicontrol(...
0129         'Parent',hpanel(3),...
0130         'Units','normalized',...
0131         'Position',[.1,.1,.8,.82],...
0132         'BackgroundColor',get(hpanel(3),'BackgroundColor'),...
0133         'HorizontalAlignment','left',...
0134         'Style','text',...
0135         'String',[...
0136         'tabpanel, by Dirk Tenne
0137         'Tab panel example, by
0138         '    Bill York
0139         'TabPanel Constructor
0140         '    v1.3, by Elmar Tarajan'
0141         'Tab Panel (Yet another
0142         '    one), by Laine Berhane'
0143         '    Kahsay
0144         'in File Exchange Site. ']);
0145
0146     uicontrol(...
0147         'Parent',hpanel(4),...
0148         'Units','normalized',...
0149         'Position',[.1,.1,.8,.76],...
0150         'BackgroundColor',get(hpanel(3),'BackgroundColor'),...
0151         'Style','text',...
0152         'String',[...
0153         '    UITABPANEL
0154         '    version 1.0
0155         '
0156         '    Created by
0157         '    Shiyong Zhao
0158         '(zhao@arch.umsi.edu)'
0159         '
0160         '    May 20, 2006 ']);
0161
0162     uicontrol(...
0163         'Parent',hstatus,...
0164         'Units','normalized',...

```



```

0165     'Position',[0,0,1,.3],...
0166     'BackgroundColor',get(hstatus,'BackgroundColor'),...
0167     'ForegroundColor',[0.6,1,1],...
0168     'HitTest','off',...
0169     'Style','text',...
0170     'String',[...
0171         ' Copyright(c) 2006 '
0172         '   Shiying Zhao   '
0173         'All rights reserved.']);
0174
0175     set(htab,'ResizeFcn',@TabResize1);
0176 end
0177
0178 %-----
0179 function CreateTab2(htab,evdt,hpanel,hstatus)
0180     uicontrol(...
0181         'Parent',hpanel(1),...
0182         'Units','normalized',...
0183         'Position',[.1,.1,.8,.8],...
0184         'BackgroundColor',get(hpanel(1),'BackgroundColor'),...
0185         'ForegroundColor','w',...
0186         'HorizontalAlignment','left',...
0187         'FontSize',12,...
0188         'Style','text',...
0189         'String',[ ...
0190             'UITABPANEL creates a group of tabbed '
0191             'panels with a consistent look and feel to'
0192             'the builtin MATLAB UI objects. '
0193             ' '
0194             'A special type of UITABPANEL is the '
0195             'popup tabpanel as shown on the left-hand '
0196             'side of this figure. ']);
0197
0198     uicontrol(...
0199         'Parent',hpanel(2),...
0200         'Units','normalized',...
0201         'Position',[.1,.1,.8,.8],...
0202         'BackgroundColor','w',...
0203         'HorizontalAlignment','left',...
0204         'Style','text',...
0205         'String',[...
0206             ' '
0207             ' The syntax of using UITABPANEL is the same as '
0208             ' UIPANEL except a few property changes. For a complete '
0209             ' description, see the online help of UITABPANEL and the '
0210             ' MATLAB script of this demo. Examples: '
0211             '
0212             '     htab = uitabpanel( 'Title',
{'A','B','C','D'} ) '
0213             '     htab = uitabpanel( 'Title',
{'A','B'},'Style','righttop' ) '
0214             '     htab = uitabpanel( 'Title',{'A','B','C'},'Position',

```

```

my_pos, '
0215         'CreateFcn',@my_fun1, 'ResizeFcn',@my_fun2) ']);
0216
0217 uicontrol(...
0218     'Parent',hpanel(3),...
0219     'Units','normalized',...
0220     'Position',[.1,.1,.8,.8],...
0221     'BackgroundColor','w',...
0222     'HorizontalAlignment','left',...
0223     'Style','text',...
0224     'String',[...
0225         '
0226         ' The following is a brief summary of some advanced features'
0227         ' of the current implementation of UITABPANEL: '
0228         ' 1. UITABPANEL can be resizable if 'ResizeFcn' is set. '
0229         ' 2. A callback routine 'SelectionChangeFcn' can be defined '
0230         ' through 'set/getappdata', which will be executed when '
0231         ' the selected tab changes. '
0232         ' 3. An empty UICONTAINER in each UITABPANEL is '
0233         ' designed mainly for a functional purpose, but is can be'
0234         ' used by users for other purposes in a number of ways. ']);
0235
0236 uicontrol(...
0237     'Parent',hpanel(4),...
0238     'Units','normalized',...
0239     'Position',[.1,.1,.8,.8],...
0240     'BackgroundColor','w',...
0241     'HorizontalAlignment','left',...
0242     'Style','text',...
0243     'String',[...
0244         '
0245         ' Although the current implementation of UITABPANEL works '
0246         ' well for small number of tabs, many improvements and new '
0247         ' features are still desirable. '
0248         '
0249         ' An immediate task is to add more styles, such as vertical'
0250         ' tabs and multi-row tabs. However, the most desirable '
0251         ' improvement to me is to rewrite UITABPANEL as a '
0252         ' MATLAB class, so that it will behave in the same way as '
0253         ' the builtin MATLAB user interface (UI) objects. ']);
0254
0255 uicontrol(...
0256     'Parent',hstatus,...
0257     'Units','normalized',...
0258     'Position',[0,0.1,1,.68],...
0259     'BackgroundColor',get(hstatus,'BackgroundColor'),...
0260     'ForegroundColor','k',...
0261     'HitTest','off',...
0262     'Style','text',...
0263     'String',['This is ',style2,' tab']);
0264 setappdata(htab,'SelectionChangeFcn',@TabStatus2);
0265

```

```

0266     set(htab, 'ResizeFcn', {@TabResize2, [0.5, 0.5]});
0267 end
0268
0269 %-----
0270 function CreateTab3(htab, evdt, hpanel, hstatus)
0271     uicontrol(...
0272         'Parent', hpanel(3), ...
0273         'Units', 'normalized', ...
0274         'Position', [.3, .1, .6, .6], ...
0275         'BackgroundColor', get(hpanel(1), 'BackgroundColor'), ...
0276         'ForegroundColor', 'w', ...
0277         'HorizontalAlignment', 'left', ...
0278         'FontSize', 12, ...
0279         'Style', 'text', ...
0280         'String', [...
0281             'Run this demo with the optional '
0282             'argument 1, 2, and 3 to view all'
0283             'implemented styles.']);
0284
0285     load('topo.mat', 'topo', 'topomap1');
0286     axes('Parent', hpanel(1));
0287     PlotTheEarth(topo, topomap1);
0288
0289     axes('Parent', hpanel(2), 'Position', [0, .1, .8, .8]);
0290     axis square off
0291     uicontrol(...
0292         'Parent', hpanel(2), ...
0293         'Units', 'normalized', ...
0294         'Position', [0.7, .4, .2, .1], ...
0295         'Style', 'pushbutton', ...
0296         'Callback', {@PlotTheGlobe1, topo}, ...
0297         'String', 'Create it!');
0298     uicontrol(...
0299         'Parent', hpanel(2), ...
0300         'Units', 'normalized', ...
0301         'Position', [0.7, .2, .2, .1], ...
0302         'Style', 'pushbutton', ...
0303         'Callback', 'cla', ...
0304         'String', 'Remove it!');
0305
0306     uicontrol(...
0307         'Parent', hstatus, ...
0308         'Units', 'normalized', ...
0309         'Position', [0, 0.1, 1, .68], ...
0310         'BackgroundColor', get(hstatus, 'BackgroundColor'), ...
0311         'ForegroundColor', 'k', ...
0312         'HitTest', 'off', ...
0313         'Style', 'text', ...
0314         'String', ['This is ', style3, ' tab']);
0315
0316     set(htab, 'ResizeFcn', {@TabResize2, [0, 0.5]});
0317 end

```

```

0318
0319 %-----
0320 function TabResize1(hobj,evdt)
0321     figpos = get(hfig,'Position');
0322     tabpos = get(hobj,'Position');
0323     tabpos(4) = figpos(4);
0324     set(hobj,'Position',tabpos);
0325
0326     width = tabpos(3)/figpos(3);
0327 end
0328
0329 %-----
0330 function TabResize2(hobj,evdt,ysiz)
0331     figpos = get(hfig,'Position');
0332     tabpos = get(hobj,'Position');
0333     tabpos([1,3]) = [width,1-width]*figpos(3)+[1,0];
0334     tabpos([2,4]) = ysiz*figpos(4)+[1,0];
0335     set(hobj,'Position',tabpos);
0336 end
0337
0338 %-----
0339 function TabStatus2(hobj,evdt)
0340     set(get(evdt.Status,'Children'),'String',['page ',num2str(evdt.NewValue)]);
0341 end
0342
0343 %-----
0344 function PlotTheEarth(topo,topomap1)
0345     contour(0:359,-89:90,topo,[0 0],'b')
0346     axis equal
0347     box on
0348     set(gca,'XLim',[0 360],'YLim',[-90 90], ...
0349         'XTick',[0 60 120 180 240 300 360], ...
0350         'YTick',[-90 -60 -30 0 30 60 90]);
0351     hold on
0352     image([0 360],[-90 90],topo,'CDataMapping','scaled');
0353     colormap(topomap1);
0354 end
0355
0356 %-----
0357 function PlotTheGlobe1(hobj,evdt,topo)
0358     [x,y,z] = sphere(50);
0359     props.AmbientStrength = 0.1;
0360     props.DiffuseStrength = 1;
0361     props.SpecularColorReflectance = .5;
0362     props.SpecularExponent = 20;
0363     props.SpecularStrength = 1;
0364     props.FaceColor= 'texture';
0365     props.EdgeColor = 'none';
0366     props.FaceLighting = 'phong';
0367     props.Cdata = topo;
0368     surface(x,y,z,props);
0369     light('position',[-1 0 1]);

```

```
0370     light('position',[-1.5 0.5 -0.5], 'color', [.6 .2 .2]);  
0371     end  
0372  
0373 %-----  
0374  
0375 end
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_fieldSelect.m](#)

cb_fieldSelect

PURPOSE

Build query

SYNOPSIS

```
function cb_fieldSelect(varargin)
```

DESCRIPTION

Build query

CROSS-REFERENCE INFORMATION

This function calls:

- [buildquery](#)

This function is called by:

SOURCE CODE

```
0001 function cb_fieldSelect(varargin)
0002
0003 % Build query
0004 buildquery();
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@specwin](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.2 2008/01/08 16:40:30 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```

```

0002 % ISFIELD tests if the given field is one of the object properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0007 %                properties.
0008 %
0009 % CALL:         isfield(a, 'field');
0010 %
0011 %                The following call returns a parameter list object that
0012 %                contains the default parameter values:
0013 %                >> pl = isfield(specwin, 'Params')
0014 %
0015 %                The following call returns the cvs version number:
0016 %                >> ver = isfield(specwin, 'Version')
0017 %
0018 % VERSION:      $Id: isfield.m,v 1.2 2008/01/08 16:40:30 ingo Exp $
0019 %
0020 % HISTORY: 25-09-07 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: isfield.m,v 1.2 2008/01/08 16:40:30 ingo Exp $';
0026
0027 % 'Params' Call
0028 if nargin == 2
0029     if isa(varargin{1}, 'specwin') && strcmp(varargin{2}, 'Params')
0030         varargout{1} = plist();
0031         return
0032     elseif isa(varargin{1}, 'specwin') && strcmp(varargin{2}, 'Version')
0033         varargout{1} = VERSION;
0034         return
0035     end
0036 end
0037
0038 as = [];
0039 field = '';
0040 for j=1:nargin
0041     if isa(varargin{j}, 'specwin')
0042         as = [as varargin{j}];
0043     end
0044     if ischar(varargin{j})
0045         field = varargin{j};
0046     end
0047 end
0048
0049 res = zeros(size(as));
0050 for j=1: numel(as)
0051     try
0052         t = as(j).(field);
0053         res(j) = 1;

```



```
0054     catch
0055         res(j) = 0;
0056     end
0057 end
0058
0059 varargout{1} = res;
0060
```

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

```

0001 function y = polydetrend(varargin)
0002 % POLYDETREND detrends the input data vector with a polynomial.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: POLYDETREND detrends the input data vector with a polynomial.
0007 %               This function is meant to be used by other ltpda_ functions.
0008 %               To work with Analysis Objects, use ltpda_polydetrend which calls
0009 %               this function.
0010 %
0011 % CALL:         y = polydetrend(x,order);
0012 %
0013 % INPUTS:       x      - vector of x values
0014 %               order - order of polynomial to fit and subtract
0015 %
0016 % OUTPUTS:      y      - detrended data
0017 %
0018 % VERSION:      $Id: polydetrend.m,v 1.2 2007/07/16 12:52:21 ingo Exp $
0019 %
0020 % HISTORY:      30-05-07 M Hewitson
0021 %               Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 if nargin == 2
0026     x      = varargin{1};
0027     order = varargin{2};
0028     t      = [1:length(x)]';
0029 elseif nargin == 3
0030     t      = varargin{1};
0031     x      = varargin{2};
0032     order = varargin{3};
0033 else
0034     error('### incorrect inputs. ');
0035 end
0036
0037 % fit polynomial
0038 p = polyfit(t, x, order);
0039
0040 % make polynomial series
0041 py = polyval(p, t);
0042
0043 % detrend
0044 y = x - py;
0045
0046 % END

```

uitabpanel

PURPOSE

UITABPANEL adds a group of tabbed uipanel container objects to a figure.

SYNOPSIS

`function hh = uitabpanel(varargin)`

DESCRIPTION

UITABPANEL adds a group of tabbed uipanel container objects to a figure. The user can click on a tab with the left mouse button to select it. One tab is always selected, and appears on top of all the orthers. The syntax is the same as UIPANEL except the following property changes:

Property	Values
-----	-----
Title	a cell array of strings specifying the title of each tab
String	(same as Title)
TitlePosition	the location of tabs in relation of the visible panel, the value is the same as UIPANEL
Style	(same as TitlePosition)
FrameBackgroundColor	the background color for the tabpanel frame
FrameBorderStyle	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
PanelBorderStyle	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](#).

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [repopgui](#) REPOGUI a GUI to interact with an LTPDA Repository.
- [uitabdemo](#) UITABDEMO This script demonstrates the use of UITABPANEL.

SUBFUNCTIONS

- [function PopupTabCbK\(hobj,evdt,indx\)](#)
- [function PopupTabSliderCbK\(hobj,evdt\)](#)
- [function PopupTabEnableAllCbK\(hobj,evdt,cstr\)](#)
- [function PopupTabResizeCbK\(hobj,evdt\)](#)
- [function TopBottomTabCbK\(hobj,evdt,indx\)](#)
- [function TopBottomTabResizeCbK\(hobj,evdt\)](#)

SOURCE CODE

```
0001 function hh = uitabpanel(varargin)
0002
0003 % UITABPANEL adds a group of tabbed uipanel container objects to a figure.
0004 % The user can click on a tab with the left mouse button to select it.
0005 % One tab is always selected, and appears on top of all the orthers.
0006 % The syntax is the same as UIPANEL except the following property changes:
0007 %
0008 %      Property              Values
0009 %      -----
0010 %      Title                 a cell array of strings specifying the title of
0011 %                           each tab
0012 %      String                (same as Title)
0013 %      TitlePosition         the location of tabs in relation of the visible
0014 %                           panel, the value is the same as UIPANEL
0015 %      Style                 (same as TitlePosition)
0016 %      FrameBackgroundColor  the background color for the tabpanel frame
0017 %      FrameBorderType       the border type for the tabpanel frame
0018 %      PanelBackgroundColor  the background color for the active tab/panel
0019 %      TitleHighlightColor   the highlight color for the active tab
0020 %      TitleForegroundColor  the foreground color for tab titiles
0021 %      TitleBackgroundColor  the background color for the inactive tabs
```

```

0022 %      SelectedItem      index specifying the currently active tab
0023 %
0024 %      A yet another type of UITABPANEL is a group of popup panels, which
0025 %      toggles open and close of a panel when its title is clicked. Additional
0026 %      properties for popup tabpanel are:
0027 %
0028 %      Property          Values
0029 %      -----
0030 %      Style              'popup'
0031 %      PanelHeights      a numeric array specifying the height in
0032 %                        characters of each popup panel
0033 %      PanelBorderType    the border type for the popup panels
0034 %      TitleHighlightColor (not applicable)
0035 %
0036 %      By defining 'CreateFcn', MATLAB will build the group of panels at
0037 %      creation of UITABPANEL. An array of handles of the UIPANELs can
0038 %      be retrieved by calling getappdata(hTabpanel,'panels').
0039 %
0040 %      Example:
0041 %      htab = uitabpanel('title',{'1','2','3'})
0042 %      hpanel = getappdata(htab,'panels')
0043 %
0044 %      See also uitabdemo, uipanel, uicontrol, set, get.
0045
0046 % Author: Shiyong Zhao (zhao@arch.umsi.edu)
0047 % Version: 1.0
0048 % First created: May 02, 2006
0049 % Last modified: May 20, 2006
0050
0051 tag = '';
0052 units = 'normalized';
0053 position = [0,0,1,1];
0054 fontweight = 'normal';
0055 [style,type] = deal('lefttop',2);
0056 visible = 'on';
0057
0058 foregroundcolor = [1,1,1];
0059 backgroundcolor = [0.7725,0.8431,0.9608];
0060 panelbackgroundcolor = [0.4314,0.5882,0.8431];
0061 titlehighlightcolor = [1,0.8,0];
0062
0063 horizontalalignment = 'center';
0064 [createfcn,deletefcn,resizefcn] = deal({}, {}, {});
0065
0066 active = 0;
0067 styles = {'popup','lefttop','righttop','leftbottom','rightbottom', ...
0068          'centertop','centerbottom'};
0069
0070 for k=1:2:nargin
0071     property = lower(varargin{k});
0072     value = varargin{k+1};
0073     switch property
0074         case {'style','tabposition','titleposition'}

```

```

0075     style = value;
0076     type = strmatch(lower(style),styles);
0077     if isempty(type)
0078         error(['Bad value for Style/TitlePosition property: ' style '.']);
0079     end
0080     case 'fontsize'
0081         fontsize = value;
0082     case {'title','string'}
0083         string = value;
0084         ntab = length(string);
0085     case {'frameposition','position'}
0086         position = value;
0087     case {'framebackgroundcolor','backgroundcolor'}
0088         backgroundcolor = value;
0089     case {'framebordertype','bordertype'}
0090         bordertype = value;
0091     case {'panelheights','heights'}
0092         panelheight = value;
0093     case {'titleforegroundcolor','foregroundcolor'}
0094         foregroundcolor = value;
0095     case {'selecteditem','active'}
0096         active = value;
0097     otherwise
0098         eval([property '= value;']);
0099     end
0100 end
0101
0102 if ~exist('parent','var')
0103     parent = gcf;
0104 end
0105 if ~exist('margins','var')
0106     if type==1
0107         margins = {[1.5,0.75,1.5,1],'characters'};
0108     else
0109         margins = {1,'pixels'};
0110     end
0111 end
0112 if ~iscell(margins)
0113     margins = {margins,units};
0114 end
0115
0116 if ~exist('bordertype','var')
0117     if type==1
0118         bordertype = 'etchedin';
0119     else
0120         bordertype = 'none';
0121     end
0122 end
0123
0124 if ~exist('titlebackgroundcolor','var')
0125     titlebackgroundcolor = 0.95*panelbackgroundcolor;
0126 end
0127

```

```

0128 cmenu = uicontextmenu;
0129 uimenu(cmenu, 'Label', 'Goto Tab');
0130 uimenu(cmenu, 'Separator', 'on');
0131
0132 htab = uipanel( ...
0133     'Parent',parent, ...
0134     'Units',units, ...
0135     'Position',position, ...
0136     'DeleteFcn',deletefcn, ...
0137     'ResizeFcn',resizefcn, ...
0138     'BackgroundColor',backgroundcolor, ...
0139     'BorderType',bordertype, ...
0140     'UIContextMenu',cmenu, ...
0141     'Visible',visible, ...
0142     'FontSize', fontsize,...
0143     'Tag',tag);
0144 set(htab,'Units','pixels');
0145 tabpos = get(htab,'Position');
0146
0147 status = uicontainer( ...
0148     'Parent',htab, ...
0149     'BackgroundColor',backgroundcolor, ...
0150     'Units',margins{2}, ...
0151     'Position',[1,1,1,1], ...
0152     'UIContextMenu',cmenu, ...
0153     'Visible','off');
0154 set(status,'Units','pixels');
0155 charsz = get(status,'Position');
0156 margins = [charsz(1:2),charsz(1:2)].*margins{1};
0157 set(status,'Units','characters');
0158 set(status,'Position',[1,1,1,1]);
0159 set(status,'Units','pixels');
0160 charsz = get(status,'Position');
0161 titleheight = 2*charsz(4);
0162
0163 if type==1
0164     %-----
0165     %  PopupTab
0166     %-----
0167     panelpos = tabpos;
0168     titlepos = [margins(1),tabpos(2)+tabpos(4)-titleheight-margins(2), ...
0169         tabpos(3)-sum(margins([1,3])),titleheight];
0170     tttextpos = [0.05,0.1,0.9,0.76];
0171
0172     if ~exist('panelheight','var')
0173         panelheight = ones(1,ntab);
0174     end
0175     if ~exist('panelbordertype','var')
0176         panelbordertype = 'line';
0177     end
0178
0179     for k=1:ntab
0180         title(k) = uicontrol( ...

```



```

0181     'Parent', htab, ...
0182     'Units', 'pixels', ...
0183     'Position', titlepos, ...
0184     'ForegroundColor', foregroundcolor, ...
0185     'BackgroundColor', titlebackgroundcolor, ...
0186     'FontWeight', fontweight, ...
0187     'FontSize', fontsize, ...
0188     'HorizontalAlignment', horizontalalignment, ...
0189     'Callback', {@PopupTabCbk, k}, ...
0190     'Style', 'toggle', ...
0191     'String', string{k}, ...
0192     'Value', 0, ...
0193     'Visible', 'on');
0194
0195     panel(k) = uipanel( ...
0196         'Parent', htab, ...
0197         'Units', 'pixels', ...
0198         'Position', titlepos+[0,0,0,panelheight(k)*charsz(4)], ...
0199         'BackgroundColor', panelbackgroundcolor, ...
0200         'BorderType', panelbordertype, ...
0201         'Visible', 'off');
0202
0203     uimenu(cmenu, ...
0204         'Label', string{k}, ...
0205         'Callback', {@PopupTabCbk, k});
0206 end
0207
0208     uimenu(cmenu, 'Separator', 'on');
0209     uimenu(cmenu, ...
0210         'Label', 'Expand All', ...
0211         'Callback', {@PopupTabEnableAllCbk, 'on'});
0212     uimenu(cmenu, ...
0213         'Label', 'Collapse All', ...
0214         'Callback', {@PopupTabEnableAllCbk, 'off'});
0215
0216     sliderpos = [tabpos(3)-14, 2, 15, tabpos(4)-3];
0217     slider = uicontrol( ...
0218         'Parent', htab, ...
0219         'Units', 'pixels', ...
0220         'Position', sliderpos, ...
0221         'BackgroundColor', panelbackgroundcolor, ...
0222         'Style', 'slider', ...
0223         'Callback', @PopupTabSliderCbk, ...
0224         'Value', 0, ...
0225         'Min', -1, ...
0226         'Max', 0, ...
0227         'Visible', 'off');
0228
0229     statuspos = titlepos;
0230     statuspos([2,4]) = [margins(4), panelpos(4)-(ntab+1)*(margins(4)+titlepos(4))];
0231     set(status, 'Position', statuspos);
0232     PopupTabCbk(title(1), [], active);

```

```

0233     set(status,'ResizeFcn',@PopupTabResizeCbK);
0234
0235 else
0236     %-----
0237     %   TopT/Bottom Tab
0238     %-----
0239     margins(3:4) = -(margins(1:2)+margins(3:4));
0240     switch type
0241         case 2 %lefttop
0242             [loop,sign] = deal(1:ntab,[1,0,1,0]);
0243         case 3 %righttop
0244             [loop,sign] = deal(ntab:-1:1,[ntab,0,0,0]);
0245         case 4 %lefttbottom
0246             [loop,sign] = deal(1:ntab,[1,0,1,1]);
0247         case 5 %rightbottom
0248             [loop,sign] = deal(ntab:-1:1,[ntab,0,0,1]);
0249         case 6 %centertop
0250             [loop,sign] = deal(1:ntab,[1,ntab,1,0]);
0251         case 7 %centerbottom
0252             [loop,sign] = deal(1:ntab,[1,ntab,1,1]);
0253     end
0254     addtitle = 2*[sign(3:4),-1,-1];
0255     if sign(3)
0256         ttextpos = [0.05,0.08,0.9,0.76];
0257     else
0258         ttextpos = [0.05,0.05,0.9,0.76];
0259     end
0260
0261     if ~exist('panelbordertype','var')
0262         panelbordertype = 'beveledout';
0263     end
0264
0265     for k=1:ntab
0266         title(k) = uipanel( ...
0267             'Parent',htab, ...
0268             'Units','pixels', ...
0269             'BackgroundColor',titlebackgroundcolor, ...
0270             'BorderType',panelbordertype, ...
0271             'ButtonDownFcn',{@TopBottomTabCbK,k}, ...
0272             'Visible','on');
0273         %titlewidth(k) = (length(string{k})+4)*1.2*charsz(3);
0274
0275         ttext(k) = uicontrol( ...
0276             'Parent',title(k), ...
0277             'Units','normalized', ...
0278             'Position',ttextpos, ...
0279             'ForegroundColor',foregroundcolor, ...
0280             'BackgroundColor',titlebackgroundcolor, ...
0281             'HorizontalAlignment',horizontalalignment, ...
0282             'Style','text', ...
0283             'String',string{k}, ...
0284             'Enable','inactive', ...
0285             'FontSize', fontsize,...

```

```

0286     'HitTest','off',...
0287     'Visible','on');
0288
0289     panel(k) = uipanel( ...
0290         'Parent',htab, ...
0291         'Units','pixels', ...
0292         'BackgroundColor',panelbackgroundcolor, ...
0293         'BorderType',panelbordertype, ...
0294         'Visible','off');
0295
0296     uimenu(cmenu, ...
0297         'Label',string{k}, ...
0298         'Callback',{@TopBottomTabCbK,k});
0299 end
0300
0301 if ~active, active = 1; end
0302 set(panel(active),'Visible','on');
0303
0304 inset(1) = uicontrol( ...
0305     'Parent',htab, ...
0306     'Units','pixels', ...
0307     'BackgroundColor',panelbackgroundcolor, ...
0308     'Style','text', ...
0309     'Visible','on');
0310 if ~isnan(titlehighlightcolor)
0311     inset(2) = uicontrol( ...
0312         'Parent',htab, ...
0313         'Units','pixels', ...
0314         'BackgroundColor',titlehighlightcolor, ...
0315         'Style','text', ...
0316         'Visible','on');
0317     inset(3) = uicontrol( ...
0318         'Parent',htab, ...
0319         'Units','pixels', ...
0320         'BackgroundColor',0.85*titlehighlightcolor, ...
0321         'Style','text', ...
0322         'Visible','on');
0323 end
0324
0325 TopBottomTabResizeCbK(status,[]);
0326 set(status,'ResizeFcn',@TopBottomTabResizeCbK);
0327 end
0328
0329 setappdata(htab,'panels',panel);
0330 setappdata(htab,'status',status);
0331 if ~isempty(createfcn)
0332     createfcn(htab,[],panel,status);
0333 end
0334
0335 if narginout, hh = htab; end
0336
0337

```

```

0338 %-----
0339 % Inner Functions for Popup Tab
0340 %-----
0341 function PopupTabCbk(hobj,evdt,indx)
0342
0343     set(title,'Value',0);
0344     if indx>0
0345         indxvis = strcmp(get(panel(indx),'Visible'),'off');
0346         if indxvis
0347             set(panel(indx),'Visible','on');
0348         else
0349             set(panel(indx),'Visible','off');
0350         end
0351     end
0352     if isempty(strmatch('on',get(panel,'Visible')))
0353         set(status,'Visible','on');
0354     else
0355         set(status,'Visible','off');
0356     end
0357
0358     currval = get(slider,'Value');
0359     if currval<0
0360         set(slider,'Value',0);
0361         PopupTabSliderCbk(slider,evdt);
0362     end
0363     titlepos = get(title(1),'Position');
0364     addtolen = titlepos(2)+titlepos(4)+margins(4);
0365
0366     for k=1:ntab
0367         titlepos = get(title(k),'Position');
0368         titlepos(2) = addtolen-titlepos(4)-margins(4);
0369         set(title(k),'Position',titlepos);
0370         addtolen = titlepos(2);
0371         visible = strcmp(get(panel(k),'Visible'),'on');
0372         if visible
0373             panelpos = get(panel(k),'Position');
0374             addtolen = addtolen-panelpos(4);
0375             panelpos(2) = addtolen;
0376             set(panel(k),'Position',panelpos);
0377         end
0378     end
0379
0380     if visible
0381         botpos = get(panel(ntab),'Position');
0382     else
0383         botpos = get(title(ntab),'Position');
0384     end
0385
0386     visible = strcmp(get(slider,'Visible'),'on');
0387     addwidth = (-1)^(~visible);
0388     if botpos(2)*addwidth>0
0389         if visible
0390             set(slider,'Visible','off');

```

```

0391     else
0392         set(slider, 'Visible', 'on');
0393     end
0394 else
0395     if visible
0396         addwidth = 0;
0397     else
0398         return;
0399     end
0400 end
0401
0402 addwidth = addwidth*sliderpos(3);
0403 for k=1:ntab
0404     titlepos = get(title(k), 'Position');
0405     titlepos(3) = titlepos(3)+addwidth;
0406     set(title(k), 'Position', titlepos);
0407     set(title(k), 'UserData', titlepos(2));
0408
0409     panelpos = get(panel(k), 'Position');
0410     panelpos(3) = panelpos(3)+addwidth;
0411     set(panel(k), 'Position', panelpos);
0412     set(panel(k), 'UserData', panelpos(2));
0413 end
0414
0415 toplen = botpos(2)-margins(2);
0416 if toplen>0
0417     set(slider, 'Value', 0);
0418     return;
0419 end
0420
0421 try
0422     if indxvis
0423         viewpos = get(panel(indx), 'Position');
0424         if viewpos(2)-viewpos(4)<0
0425             if viewpos(4)+margins(4)<=tabpos(4);
0426                 currval = min(0, viewpos(2)-margins(4)/2);
0427             else
0428                 viewpos = get(title(indx), 'Position');
0429                 currval = viewpos(2)+viewpos(4)+margins(4)/2-tabpos(4);
0430             end
0431         end
0432     end
0433 catch
0434 end
0435
0436 set(slider, 'Min', toplen);
0437 set(slider, 'Value', max(currval, toplen));
0438 set(slider, 'SliderStep', [0.3, 1]/max(1, -toplen/100));
0439 PopupTabSliderCbk(slider, evdt);
0440
0441 end
0442
0443

```

```

0444 %-----
0445 function PopupTabSliderCbk(hobj,evdt)
0446
0447     currval = get(hobj,'Value');
0448     for k=1:ntab
0449         titlepos = get(title(k),'Position');
0450         titlepos(2) = get(title(k),'UserData')-currval;
0451         set(title(k),'Position',titlepos);
0452         panelpos = get(panel(k),'Position');
0453         panelpos(2) = get(panel(k),'UserData')-currval;
0454         set(panel(k),'Position',panelpos);
0455     end
0456
0457 end
0458
0459
0460 %-----
0461 function PopupTabEnableAllCbk(hobj,evdt,cstr)
0462
0463     set(panel,'Visible',cstr);
0464     PopupTabCbk(hobj,evdt,0);
0465
0466 end
0467
0468
0469 %-----
0470 function PopupTabResizeCbk(hobj,evdt)
0471
0472     oldpos = tabpos;
0473     units = get(htab,'Units');
0474     set(htab,'Units','pixels');
0475     tabpos = get(htab,'Position');
0476     set(htab,'Units',units);
0477
0478     addtopos = tabpos-oldpos;
0479     sliderpos([1,2,4]) = sliderpos([1,2,4])+addtopos([1,2,4]);
0480     set(slider,'Position',sliderpos);
0481
0482     addtopos = [tabpos(3:4)-oldpos(3:4),0,0];
0483     currval = get(slider,'Value');
0484     if currval<0
0485         set(slider,'Value',0);
0486         PopupTabSliderCbk(slider,evdt);
0487     end
0488
0489     for k=1:ntab
0490         titlepos = get(title(k),'Position');
0491         set(title(k),'Position',titlepos+addtopos);
0492     end
0493     PopupTabCbk(title(1),evdt,0);
0494
0495     visible = strcmp(get(slider,'Visible'),'on');

```

```

0496     if visible & currval<0
0497         set(slider, 'Value', max(currval, get(slider, 'Min')));
0498         PopupTabSliderCbk(slider, evdt);
0499     end
0500
0501 end
0502
0503
0504 %-----
0505 % Inner Functions for Top/Bottom Tab
0506 %-----
0507 function TopBottomTabCbk(hobj, evdt, indx)
0508
0509     if active==indx, return; end
0510
0511     % deactivate tab(active)
0512     n = active;
0513     titlepos = get(title(n), 'Position');
0514     titlepos = titlepos+addtitle.*[n==sign(1), 1, n==sign(1), 1];
0515     titlepos(3) = titlepos(3)+addtitle(3)*(n==sign(2));
0516     set(title(n), 'Position', titlepos, 'BackgroundColor', titlebackgroundcolor);
0517     set(ttext(n), 'FontWeight', 'normal', 'BackgroundColor', titlebackgroundcolor);
0518     set(panel(n), 'Visible', 'off');
0519
0520     % activate tab(indx)
0521     active = indx;
0522     titlepos = get(title(indx), 'Position');
0523     titlepos = titlepos-addtitle.*[indx==sign(1), 1, indx==sign(1), 1];
0524     titlepos(3) = titlepos(3)-addtitle(3)*(indx==sign(2));
0525
0526     set(title(indx), 'Position', titlepos, 'BackgroundColor', panelbackgroundcolor);
0527     set(ttext(indx), 'FontWeight', 'bold', 'BackgroundColor', panelbackgroundcolor);
0528     insetpos = [titlepos(1)+1, titlepos(2)+sign(4)*titlepos(4)-2, titlepos(3)-
0529 1, 3];
0529     set(inset(1), 'Position', insetpos);
0530     try
0531         insetpos = insetpos+[0, (1-2*sign(4))*titleheight+2*sign(4), 0, -2];
0532         set(inset(2), 'Position', insetpos);
0533         set(inset(3), 'Position', insetpos+[2, 1-2*sign(4), -4, 0]);
0534     catch
0535     end
0536     set(panel(indx), 'Visible', 'on');
0537
0538     if isappdata(htab, 'SelectionChangeFcn')
0539         SelectionChangeFcn = getappdata(htab, 'SelectionChangeFcn');
0540         eventdata = struct('EventName', 'SelectionChanged', ...
0541             'OldValue', n, 'NewValue', indx, 'Panels', panel, 'Status', status);
0542         if iscell(SelectionChangeFcn)
0543             SelectionChangeFcn{1}(hobj, eventdata, SelectionChangeFcn{2:end});
0544         else
0545             SelectionChangeFcn(hobj, eventdata);
0546         end
0547     end

```



```

0548
0549     end
0550
0551
0552 %-----
0553     function TopBottomTabResizeCbk(hobj,evdt)
0554
0555         units = get(htab,'Units');
0556         set(htab,'Units','pixels');
0557         tabpos = get(htab,'Position');
0558         set(htab,'Units',units);
0559
0560         titlepos = [0,0,0,titleheight];
0561         if sign(4)
0562             panelpos = [0,titleheight+1,tabpos(3),tabpos(4)-titleheight]+margins;
0563             titlepos(2) = panelpos(2)-titlepos(4);
0564         else
0565             panelpos = [0,0,tabpos(3),tabpos(4)-titleheight]+margins;
0566             titlepos(2) = panelpos(2)+panelpos(4);
0567         end
0568         set(panel,'Position',panelpos);
0569
0570         n = sum(cellfun('length',string))+2*ntab;
0571         if type<6
0572             if (n+2*ntab)*1.2*charsz(3)+2>panelpos(3)
0573                 [loop,sign(1:3)] = deal(1:ntab,[1,ntab,1]);
0574             else
0575                 if type==2 | type==4
0576                     [loop,sign(1:3)] = deal(1:ntab,[1,0,1]);
0577                 else
0578                     [loop,sign(1:3)] = deal(ntab:-1:1,[ntab,0,0]);
0579                 end
0580             end
0581             addtitle = 2*[sign(3:4),-1,-1];
0582         end
0583         titlepos(1) = panelpos(1)+(1-sign(3))*panelpos(3);
0584         for k=loop
0585             if sign(2)
0586                 titlepos(3) = panelpos(3)*(length(string{k})+2)/n;
0587                 addtopos = addtitle.*[k==1,1,k==1|k==ntab,1];
0588             else
0589                 titlepos(3) = (length(string{k})+4)*1.2*charsz(3);
0590                 addtopos = addtitle.*[k==sign(1),1,k==sign(1),1];
0591             end
0592             titlepos(1) = titlepos(1)-(1-sign(3))*titlepos(3);
0593             set(title(k),'Position',titlepos+addtopos);
0594             titlepos(1) = titlepos(1)+sign(3)*titlepos(3);
0595         end
0596
0597         statuspos = titlepos;
0598         statuspos(1) = sign(3)*titlepos(1)+2;
0599         statuspos(3) = titlepos(1)+sign(3)*(tabpos(3)-2*titlepos(1))-4;
0600

```



```

0601     titlepos = get(title(active), 'Position');
0602     titlepos = titlepos-addtitle.*[active==sign(1),1,active==sign(1),1];
0603     titlepos(3) = titlepos(3)-addtitle(3)*(active==sign(2));
0604     set(title(active), 'Position', titlepos, 'BackgroundColor',
panelbackgroundcolor);
0605     set(ttext(active), 'FontWeight', 'bold', 'BackgroundColor',
panelbackgroundcolor);
0606
0607     insetpos = [titlepos(1)+1,titlepos(2)+sign(4)*titlepos(4)-2,titlepos(3)-
1,3];
0608     set(inset(1), 'Position', insetpos);
0609     try
0610         insetpos = insetpos+[0,(1-2*sign(4))*titleheight+2*sign(4),0,-2];
0611         set(inset(2), 'Position', insetpos);
0612         set(inset(3), 'Position', insetpos+[2,1-2*sign(4),-4,0]);
0613     catch
0614     end
0615     try
0616         % This causes the above statements being excuted twice for each resizing.
0617         set(status, 'Position', statuspos, 'Visible', 'on');
0618     catch
0619         set(status, 'Visible', 'off');
0620     end
0621
0622 end
0623
0624 end

```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_getDBs.m](#)

cb_getDBs

PURPOSE

SYNOPSIS

```
function cb_getDBs(varargin)
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function cb_getDBs(varargin)
0002
0003 myh      = varargin{1};
0004 mainfig = varargin{4};
0005
0006 tables = {'collections', 'objmeta', 'objs', 'transactions', 'users'};
0007
0008 % Get conn
0009 conn = getappdata(mainfig, 'connection');
0010
0011 if isempty(conn)
0012
0013     % get hostname
0014     hostname = get(findobj('Tag','serverTxt'), 'String');
0015
0016     % connect
0017     [conn, username] = mysql_connect(hostname);
0018     if ~isa(conn, 'database')
0019         error('# couldn't connect to server');
```

```

0020     end
0021 end
0022
0023 % get the databases
0024 q = 'show databases';
0025 curs = exec(conn, q);
0026 curs = fetch(curs);
0027
0028 ltpda_dbs = [];
0029 % Now get the LTPDA databases
0030 for j=1:length(curs.Data)
0031     db = curs.Data{j};
0032
0033     if ~strcmp(db, 'information_schema')
0034
0035         q = sprintf('show tables from %s', db);
0036         c = exec(conn, q);
0037         c = fetch(c);
0038         match = 0;
0039         for l=1:length(c.Data)
0040             for k=1:length(c.Data{l})
0041                 if strcmp(c.Data{l}(k), tables{k})
0042                     match = match + 1;
0043                 end
0044             end
0045         end
0046         close(c);
0047         if match == 5
0048             % we have an LTPDA database
0049             ltpda_dbs = [ltpda_dbs cellstr(db)];
0050         end
0051     end
0052
0053 end
0054 close(curs);
0055
0056 % Set the list
0057 id = findobj('Tag', 'dbSelect');
0058 set(id, 'String', ltpda_dbs, 'Value', 1);
0059
0060
0061
0062

```

[Home](#) > [classes](#) > [@time](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.2 2008/01/08 16:39:46 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```

```

0002 % ISFIELD tests if the given field is one of the object properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0007 %                properties.
0008 %
0009 % CALL:         isfield(a, 'field');
0010 %
0011 %                The following call returns a parameter list object that
0012 %                contains the default parameter values:
0013 %                >> pl = isfield(time, 'Params')
0014 %
0015 %                The following call returns the cvs version number:
0016 %                >> ver = isfield(time, 'Version')
0017 %
0018 % VERSION:       $Id: isfield.m,v 1.2 2008/01/08 16:39:46 ingo Exp $
0019 %
0020 % HISTORY: 25-09-07 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: isfield.m,v 1.2 2008/01/08 16:39:46 ingo Exp $';
0026
0027 % 'Params' Call
0028 if nargin == 2
0029     if isa(varargin{1}, 'time') && strcmp(varargin{2}, 'Params')
0030         varargout{1} = plist();
0031         return
0032     elseif isa(varargin{1}, 'time') && strcmp(varargin{2}, 'Version')
0033         varargout{1} = VERSION;
0034         return
0035     end
0036 end
0037
0038 as = [];
0039 field = '';
0040 for j=1:nargin
0041     if isa(varargin{j}, 'time')
0042         as = [as varargin{j}];
0043     end
0044     if ischar(varargin{j})
0045         field = varargin{j};
0046     end
0047 end
0048
0049 res = zeros(size(as));
0050 for j=1: numel(as)
0051     try
0052         t = as(j).(field);
0053         res(j) = 1;

```

```
0054     catch
0055         res(j) = 0;
0056     end
0057 end
0058
0059 varargout{1} = res;
0060
```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_mainfigClose.m](#)

cb_mainfigClose

PURPOSE [^](#)

SYNOPSIS [^](#)

```
function cb_mainfigClose(varargin)
```

DESCRIPTION [^](#)

CROSS-REFERENCE INFORMATION [^](#)

This function calls:

This function is called by:

SOURCE CODE [^](#)

```
0001 function cb_mainfigClose(varargin)
0002
0003 disp('* Goodbye from the LTPDA Repository GUI *')
0004 delete(varargin{1})
0005
0006
0007
0008
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@timeformat](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.2 2008/01/08 16:39:46 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```



```

0002 % ISFIELD tests if the given field is one of the object properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0007 %                properties.
0008 %
0009 % CALL:         isfield(a, 'field');
0010 %
0011 %                The following call returns a parameter list object that
0012 %                contains the default parameter values:
0013 %                >> pl = isfield(timeformat, 'Params')
0014 %
0015 %                The following call returns the cvs version number:
0016 %                >> ver = isfield(timeformat, 'Version')
0017 %
0018 % VERSION:      $Id: isfield.m,v 1.2 2008/01/08 16:39:46 ingo Exp $
0019 %
0020 % HISTORY: 25-09-07 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: isfield.m,v 1.2 2008/01/08 16:39:46 ingo Exp $';
0026
0027 % 'Params' Call
0028 if nargin == 2
0029     if isa(varargin{1}, 'timeformat') && strcmp(varargin{2}, 'Params')
0030         varargout{1} = plist();
0031         return
0032     elseif isa(varargin{1}, 'timeformat') && strcmp(varargin{2}, 'Version')
0033         varargout{1} = VERSION;
0034         return
0035     end
0036 end
0037
0038 as = [];
0039 field = '';
0040 for j=1:nargin
0041     if isa(varargin{j}, 'timeformat')
0042         as = [as varargin{j}];
0043     end
0044     if ischar(varargin{j})
0045         field = varargin{j};
0046     end
0047 end
0048
0049 res = zeros(size(as));
0050 for j=1: numel(as)
0051     try
0052         t = as(j).(field);
0053         res(j) = 1;

```

```
0054     catch
0055         res(j) = 0;
0056     end
0057 end
0058
0059 varargout{1} = res;
0060
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_refreshObjsList.m](#)

cb_refreshObjsList

PURPOSE

get a list of objects

SYNOPSIS

```
function cb_refreshObjsList(varargin)
```

DESCRIPTION

get a list of objects

CROSS-REFERENCE INFORMATION

This function calls:

- [getWorkspaceObjs](#) get base workspace variables
- [setWorkspaceObjsList](#)

This function is called by:

SOURCE CODE

```
0001 function cb_refreshObjsList(varargin)
0002
0003 % get a list of objects
0004 objs = getWorkspaceObjs();
0005
0006 % set list
0007 setWorkspaceObjsList(objs);
0008
0009
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@timespan](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.2 2008/01/08 16:39:46 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION [^](#)

This function calls:

This function is called by:

SOURCE CODE [^](#)

```
0001 function varargout = isfield(varargin)
```

```

0002 % ISFIELD tests if the given field is one of the object properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0007 %                properties.
0008 %
0009 % CALL:         isfield(a, 'field');
0010 %
0011 %                The following call returns a parameter list object that
0012 %                contains the default parameter values:
0013 %                >> pl = isfield(timespan, 'Params')
0014 %
0015 %                The following call returns the cvs version number:
0016 %                >> ver = isfield(timespan, 'Version')
0017 %
0018 % VERSION:      $Id: isfield.m,v 1.2 2008/01/08 16:39:46 ingo Exp $
0019 %
0020 % HISTORY: 25-09-07 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: isfield.m,v 1.2 2008/01/08 16:39:46 ingo Exp $';
0026
0027 % 'Params' Call
0028 if nargin == 2
0029     if isa(varargin{1}, 'timespan') && strcmp(varargin{2}, 'Params')
0030         varargout{1} = plist();
0031         return
0032     elseif isa(varargin{1}, 'timespan') && strcmp(varargin{2}, 'Version')
0033         varargout{1} = VERSION;
0034         return
0035     end
0036 end
0037
0038 as = [];
0039 field = '';
0040 for j=1:nargin
0041     if isa(varargin{j}, 'timespan')
0042         as = [as varargin{j}];
0043     end
0044     if ischar(varargin{j})
0045         field = varargin{j};
0046     end
0047 end
0048
0049 res = zeros(size(as));
0050 for j=1: numel(as)
0051     try
0052         t = as(j).(field);
0053         res(j) = 1;

```

```
0054     catch
0055         res(j) = 0;
0056     end
0057 end
0058
0059 varargout{1} = res;
0060
```

[Home](#) > [classes](#) > [@pole](#) > [private](#) > pri2fq.m

pri2fq

PURPOSE

Convert complex pole/zero into frequency/Q pole/zero representation.

SYNOPSIS

```
function [f0, q]= pri2fq(c)
```

DESCRIPTION

Convert complex 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

```
0001 function [f0, q]= pri2fq(c)
0002
0003 % Convert complex pole/zero into frequency/Q pole/zero representation.
0004 %
0005 % [f0, q]= pri2fq(c)
0006 %
0007 % M Hewitson 2003-03-12
0008 %
0009 %
0010 % $Id: pri2fq.m,v 1.3 2008/01/17 16:00:06 anneke Exp $
```



```
0011 %
0012
0013
0014 if(nargin==0)
0015     disp('usage: [f0, q]= ri2fq(c)');
0016     return
0017 end
0018
0019 a = real(c(1));
0020 b = imag(c(1));
0021
0022 % f0 = -a*sqrt(b^2/a^2 + 1) / 2 / pi;
0023 % q = sqrt(b^2/a^2 +1)/2;
0024
0025 k = -a*sqrt(b^2/a^2 + 1) * 2 * pi;
0026 f0 = 1/k;
0027 q = sqrt(b^2/a^2 +1)/2;
0028
0029 % END
```

[Home](#) > [classes](#) > [@ao](#) > [testing](#) > vertcat.m

vertcat

PURPOSE

VERTCAT overload horizontal concatenation [] for AOs.

SYNOPSIS

function b = vertcat(varargin)

DESCRIPTION

VERTCAT overload horizontal concatenation [] for AOs.

usage: b = [a1;a2];

M Hewitson 02-05-07

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function b = vertcat(varargin)
0002
0003 % VERTCAT overload horizontal concatenation [] for AOs.
0004 %
0005 % usage: b = [a1;a2];
0006 %
0007 % M Hewitson 02-05-07
0008 %
0009
```

```
0010 ALGONAME = mfilename;
0011 VERSION   = '$Id:$';
0012
0013 ai = [];
0014 for j=1:nargin
0015     if isa(varargin{j}, 'ao')
0016         ai = [ai varargin{j}];
0017     end
0018 end
0019
0020 ni = length(ai);
0021
0022 % Make outputs
0023 b = [];
0024 for j=1:ni
0025     a = ai(j);
0026     h = history('index', VERSION, [], get(a, 'hist'));
0027     % make output analysis object
0028     aout = ao(a.data, h);
0029     % set name
0030     aout = set(aout, 'name', get(a, 'name'));
0031     b = [b;aout];
0032
0033 end
0034
0035
0036 % END
```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > cb_refreshObjsListQL.m

cb_refreshObjsListQL

PURPOSE

get a list of objects

SYNOPSIS

```
function cb_refreshObjsListQL(varargin)
```

DESCRIPTION

get a list of objects

CROSS-REFERENCE INFORMATION

This function calls:

- [getWorkspaceObjs](#) get base workspace variables
- [setWorkspaceObjsListQL](#)

This function is called by:

SOURCE CODE

```
0001 function cb_refreshObjsListQL(varargin)
0002
0003 % get a list of objects
0004 objs = getWorkspaceObjs();
0005
0006 % set list
0007 setWorkspaceObjsListQL(objs);
0008
0009
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@tsdata](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.2 2007/11/14 18:18:37 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```

```

0002 % ISFIELD tests if the given field is one of the object properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0007 %                properties.
0008 %
0009 % CALL:         isfield(a, 'field');
0010 %
0011 %                The following call returns a parameter list object that
0012 %                contains the default parameter values:
0013 %                >> pl = isfield(tsddata, 'Params')
0014 %
0015 %                The following call returns the cvs version number:
0016 %                >> ver = isfield(tsddata, 'Version')
0017 %
0018 % VERSION:      $Id: isfield.m,v 1.2 2007/11/14 18:18:37 ingo Exp $
0019 %
0020 % HISTORY: 25-09-07 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: isfield.m,v 1.2 2007/11/14 18:18:37 ingo Exp $';
0026
0027 % 'Params' Call
0028 if nargin == 2
0029     if isa(varargin{1}, 'tsdata') && strcmp(varargin{2}, 'Params')
0030         varargout{1} = plist();
0031         return
0032     elseif isa(varargin{1}, 'tsdata') && strcmp(varargin{2}, 'Version')
0033         varargout{1} = VERSION;
0034         return
0035     end
0036 end
0037
0038 as = [];
0039 field = '';
0040 for j=1:nargin
0041     if isa(varargin{j}, 'tsdata')
0042         as = [as varargin{j}];
0043     end
0044     if ischar(varargin{j})
0045         field = varargin{j};
0046     end
0047 end
0048
0049 res = zeros(size(as));
0050 for j=1: numel(as)
0051     try
0052         t = as(j).(field);
0053         res(j) = 1;

```

```
0054     catch
0055         res(j) = 0;
0056     end
0057 end
0058
0059 varargout{1} = res;
0060
```


[Home](#) > [classes](#) > [@zero](#) > [private](#) > pri2fq.m

pri2fq

PURPOSE

Convert complex pole/zero into frequency/Q pole/zero representation.

SYNOPSIS

```
function [f0, q]= pri2fq(c)
```

DESCRIPTION

Convert complex 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

```
0001 function [f0, q]= pri2fq(c)
0002
0003 % Convert complex pole/zero into frequency/Q pole/zero representation.
0004 %
0005 % [f0, q]= pri2fq(c)
0006 %
0007 % M Hewitson 2003-03-12
0008 %
0009 %
0010 % $Id: pri2fq.m,v 1.3 2008/01/17 15:59:42 anneke Exp $
```

```
0011 %
0012
0013
0014 if(nargin==0)
0015     disp('usage: [f0, q]= ri2fq(c)');
0016     return
0017 end
0018
0019 a = real(c(1));
0020 b = imag(c(1));
0021
0022 k = -a*sqrt(b^2/a^2 + 1) * 2 * pi;
0023 f0 = 1/k;
0024 q = sqrt(b^2/a^2 +1)/2;
0025
0026
0027
0028 % f0 = -a*sqrt(b^2/a^2 + 1) / 2 / pi;
0029 % END
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_BH92.m

win_BH92

PURPOSE

WIN_BH92 return BH92 window sample for input z.

SYNOPSIS

function w = win_BH92(z)

DESCRIPTION

WIN_BH92 return BH92 window sample for input z.

M Hewitson 18-05-07

\$Id: win_BH92.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_BH92(z)
0002
0003 % WIN_BH92 return BH92 window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_BH92.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.35875 - 0.48829 * cos (z) + 0.14128 * cos (2 * z) - 0.01168 * cos (3 * z);
0012
0013 % Make window struct
0014 w.name      = 'BH92';
0015 w.alpha     = 0;
0016 w.psll      = 92;
0017 w.rov       = 66.1;
0018 w.nenbw     = 2.0044;
0019 w.w3db      = 1.8962;
```

```
0020 w.flatness = -0.8256;  
0021 w.ws       = sum(v);  
0022 w.ws2      = sum(v.*v);  
0023 w.win      = v;  
0024  
0025 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_retrieveBtn.m](#)

cb_retrieveBtn

PURPOSE

Get connection

SYNOPSIS

```
function cb_retrieveBtn(varargin)
```

DESCRIPTION

Get connection

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SUBFUNCTIONS

- [function \[ids, cids\] = getIds\(\)](#)

SOURCE CODE

```
0001 function cb_retrieveBtn(varargin)
0002
0003 % Get connection
0004 mainfig = findobj('Tag', 'LTPDARepomainfig');
0005 conn    = getappdata(mainfig, 'connection');
0006 if isempty(conn) || ~isa(conn, 'database')
0007     errordlg('Please connect to a database first', 'No connection found');
0008     return
0009 end
0010
0011 % Get variable name specs
0012 h = findobj('Tag', 'objPrefixTxt');
0013 prefix = get(h, 'String');
```

```

0014 if isempty(prefix)
0015     prefix = 'obj';
0016     warning('! Using default prefix (obj)');
0017 end
0018 h = findobj('Tag', 'appendObjTypeChk');
0019 appendObj = get(h, 'Value');
0020
0021 % Get IDs from text box
0022 [ids, cids] = getIds()
0023
0024
0025 %-----
0026 % Retrieve these ids
0027 objs = [];
0028 for j=1:length(ids)
0029     disp(sprintf('+ retrieving object %d', ids(j)));
0030     tt = mysql_getObjType(conn, ids(j));
0031     objname = sprintf('%s%03d', prefix, ids(j));
0032     if appendObj
0033         objname = [objname '_' tt];
0034     end
0035     cmd = sprintf('obj = %s(conn, ids(j));', tt);
0036     eval(cmd);
0037
0038     assignin('base', objname, obj);
0039
0040     if j==1
0041         objs = {obj};
0042     else
0043         objs = [objs {obj}];
0044     end
0045 end
0046
0047 disp(sprintf('** Retrieved %d objects.', length(ids)));
0048
0049 %-----
0050 % Retrieve these Collections
0051
0052 for k=1:length(cids)
0053
0054     % get Ids from Cid
0055     ids = mysql_getObjIds(conn, cids(k));
0056
0057     for j=1:length(ids)
0058         disp(sprintf('+ retrieving object %d', ids(j)));
0059         tt = mysql_getObjType(conn, ids(j));
0060         objname = sprintf('%sC%03d_%03d', prefix, cids(k), ids(j));
0061         if appendObj

```

```

0062     objname = [objname '_' tt];
0063     end
0064     cmd = sprintf('obj = %s(conn, ids(j));', tt);
0065     eval(cmd);
0066
0067     assignin('base', objname, obj);
0068
0069     if j==1
0070         objs = {obj};
0071     else
0072         objs = [objs {obj}];
0073     end
0074 end
0075 end
0076
0077 disp(sprintf('** Retrieved %d objects.', length(ids)));
0078
0079
0080 end
0081
0082
0083 function [ids, cids] = getIds()
0084
0085 ids = [];
0086 cids = [];
0087
0088 th = findobj('Tag', 'retrieveIDsTxt');
0089 idStr = get(th, 'String');
0090 cs = cellstr(idStr);
0091
0092 for j=1:length(cs)
0093     disp('-----')
0094     ls = cs{j};
0095     [s,r] = strtok(ls);
0096     if ~isempty(s)
0097         if s(1) == 'c'
0098             s = s(2:end);
0099             cids = [cids round(str2num(s))];
0100         else
0101             ids = [ids round(str2num(s))];
0102         end
0103     end
0104     while ~isempty(r)
0105         [s,r] = strtok(r);
0106         if ~isempty(s)
0107             if s(1) == 'c'
0108                 s = s(2:end);
0109                 cids = [cids round(str2num(s))];

```

```
0110         else
0111             ids = [ids round(str2num(s))];
0112         end
0113     end
0114 end
0115 end
0116
0117 end
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@xydata](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.2 2007/11/14 18:18:37 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```

```

0002 % ISFIELD tests if the given field is one of the object properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0007 %                properties.
0008 %
0009 % CALL:         isfield(a, 'field');
0010 %
0011 %                The following call returns a parameter list object that
0012 %                contains the default parameter values:
0013 %                >> pl = isfield(xydata, 'Params')
0014 %
0015 %                The following call returns the cvs version number:
0016 %                >> ver = isfield(xydata, 'Version')
0017 %
0018 % VERSION:      $Id: isfield.m,v 1.2 2007/11/14 18:18:37 ingo Exp $
0019 %
0020 % HISTORY: 25-09-07 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: isfield.m,v 1.2 2007/11/14 18:18:37 ingo Exp $';
0026
0027 % 'Params' Call
0028 if nargin == 2
0029     if isa(varargin{1}, 'xydata') && strcmp(varargin{2}, 'Params')
0030         varargout{1} = plist();
0031         return
0032     elseif isa(varargin{1}, 'xydata') && strcmp(varargin{2}, 'Version')
0033         varargout{1} = VERSION;
0034         return
0035     end
0036 end
0037
0038 as = [];
0039 field = '';
0040 for j=1:nargin
0041     if isa(varargin{j}, 'xydata')
0042         as = [as varargin{j}];
0043     end
0044     if ischar(varargin{j})
0045         field = varargin{j};
0046     end
0047 end
0048
0049 res = zeros(size(as));
0050 for j=1: numel(as)
0051     try
0052         t = as(j).(field);
0053         res(j) = 1;

```

```
0054     catch
0055         res(j) = 0;
0056     end
0057 end
0058
0059 varargout{1} = res;
0060
```

[Home](#) > [classes](#) > [@provenance](#) > provenance.m

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 1.16 2008/01/07 13:54:11 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.
- [xmlparse](#) XMLPARSE creates a provenance from a provenance xml file.

This function is called by:

- [provenance](#) PROVENANCE constructors for provenance class.

SOURCE CODE

```
0001 function p = provenance(varargin)
0002 % PROVENANCE constructors for provenance class.
```

```

0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION:  PROVENANCE constructors for provenance class.
0007 %
0008 % CONSTRUCTORS: p = provenance();
0009 %                 p = provenance('creator');
0010 %
0011 % VERSION:       $Id: provenance.m,v 1.16 2008/01/07 13:54:11 ingo Exp $
0012 %
0013 % HISTORY:       07-05-2007 M Hewitson
0014 %                 Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: provenance.m,v 1.16 2008/01/07 13:54:11 ingo Exp $';
0019 creator = [];
0020
0021 %%%% Check if this is a call for parameters %%%%
0022 if nargin == 2
0023     if isa(varargin{1}, 'provenance') && ischar(varargin{2})
0024         in = char(varargin{2});
0025         if strcmp(in, 'Params')
0026             p = plist;
0027             return
0028         elseif strcmp(in, 'Version')
0029             p = VERSION;
0030             return
0031         end
0032     end
0033 end
0034
0035 %%%%%%%%% From DATABASE %%%%%%%%%
0036 if nargin == 2
0037     if isa(varargin{1}, 'database')
0038         p = retrieve(varargin{1}, varargin{2:end});
0039         return
0040     else
0041         error('### Incorrect provenance constructor. ');
0042     end
0043
0044 elseif nargin == 1
0045
0046     %%%%%%%%% Create from XML fragment %%%%%%%%%
0047     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0048         p = fromxml(varargin{1});
0049         return
0050     %%%%%%%%% p = provenance(provenance) %%%%%%%%%
0051     elseif isa(varargin{1}, 'provenance')
0052         p = varargin{1};
0053         return
0054     %%%%%%%%% From File %%%%%%%%%

```

```

0055     elseif ischar(varargin{1})
0056
0057         filename = varargin{1};
0058         [path, name, ext, vers] = fileparts(filename);
0059
0060         if ~isempty(ext)
0061             switch ext
0062                 case '.mat'
0063                     p = load(filename);
0064                     p = p.a;
0065                 case '.xml'
0066                     p = xmlparse(provenance, filename);
0067                 otherwise
0068                     error('### Unknown file type.');
```

```

0069             end
0070             return
0071         else
0072             % Set creator
0073             creator = varargin{1};
0074         end
0075         %%%%%%%%% Create from structure %%%%%%%%%
0076         elseif isstruct(varargin{1})
0077
0078             p = class(varargin{1}, 'provenance');
0079             return
0080         else
0081             error('### incorrect inputs');
0082         end
0083     end
0084     created = time;
0085
0086     %-----
0087     % User Info
0088     %
0089     try
0090         if isempty(creator)
0091             creator = char(java.lang.System.getProperties.getProperty('user.name'));
0092         end
0093     catch
0094         creator = 'unknown';
0095     end
0096
0097     %-----
0098     % Machine Info
0099     %
0100     ipaddress = char(getHostAddress(java.net.InetAddress.getLocalHost));
0101     hostname = char(getHostName(java.net.InetAddress.getLocalHost));
0102     os = computer;
0103
0104     %-----
0105     % Process software versions
0106     %

```

```
0107
0108 matlab_version = getappdata(0, 'matlab_version');
0109 sigproc_version = getappdata(0, 'sigproc_version');
0110 ltpda_version   = getappdata(0, 'ltpda_version');
0111
0112 %-----
0113 % Make provenance object
0114 %
0115
0116 p.creator      = creator;
0117 p.created      = created;
0118 p.ip           = ipaddress;
0119 p.hostname     = hostname;
0120 p.os           = os;
0121 p.matlab_version = matlab_version;
0122 p.sigproc_version = sigproc_version;
0123 p.ltpda_version  = ltpda_version;
0124 p = class(p, 'provenance');
0125
0126 % END
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_Bartlett.m

win_Bartlett

PURPOSE

WIN_BARTLETT return Bartlett window sample for input z.

SYNOPSIS

function w = win_Bartlett(z)

DESCRIPTION

WIN_BARTLETT return Bartlett window sample for input z.

M Hewitson 18-05-07

\$Id: win_Bartlett.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_Bartlett(z)
0002
0003 % WIN_BARTLETT return Bartlett window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Bartlett.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```



```
0010
0011 v = z * 2;
0012 idx = find(v>1);
0013 v(idx) = 2-v(idx);
0014
0015
0016 % Make window struct
0017 w.name      = 'Bartlett';
0018 w.alpha     = 0;
0019 w.psll      = 26.5;
0020 w.rov       = 50;
0021 w.nenbw     = 1.3333;
0022 w.w3db      = 1.2736;
0023 w.flatness  = -1.8242;
0024 w.ws        = sum(v);
0025 w.ws2       = sum(v.*v);
0026 w.win       = v;
0027
0028
0029
0030 % END
```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_serverSelect.m](#)

cb_serverSelect

PURPOSE [^](#)

SYNOPSIS [^](#)

```
function cb_serverSelect(varargin)
```

DESCRIPTION [^](#)

CROSS-REFERENCE INFORMATION [^](#)

This function calls:

This function is called by:

SOURCE CODE [^](#)

```
0001 function cb_serverSelect(varargin)
0002
0003 myh = varargin{1};
0004
0005 mainfig = findobj('Tag', 'LTPDARepomainfig');
0006
0007 % Get server string
0008 servers = get(myh, 'String');
0009 sid      = get(myh, 'Value');
0010
0011 server = servers{sid};
0012
0013 % Set server text field
0014
0015 stxth = findobj('Tag', 'serverTxt');
0016 set(stxth, 'String', server);
0017
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@xyzdata](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.1 2007/12/24 19:47:06 hewitson Exp \$

HISTORY: 24-12-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```

```

0002 % ISFIELD tests if the given field is one of the object properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0007 %                properties.
0008 %
0009 % CALL:         isfield(a, 'field');
0010 %
0011 %                The following call returns a parameter list object that
0012 %                contains the default parameter values:
0013 %                >> pl = isfield(xyzdata, 'Params')
0014 %
0015 %                The following call returns the cvs version number:
0016 %                >> ver = isfield(xyzdata, 'Version')
0017 %
0018 % VERSION:       $Id: isfield.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $
0019 %
0020 % HISTORY: 24-12-07 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: isfield.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $';
0026
0027 % 'Params' Call
0028 if nargin == 2
0029     if strcmp(varargin{2}, 'Params')
0030         varargout{1} = plist();
0031         return
0032     elseif strcmp(varargin{2}, 'Version')
0033         varargout{1} = VERSION;
0034         return
0035     end
0036 end
0037
0038 as = [];
0039 field = '';
0040 for j=1:nargin
0041     if isa(varargin{j}, 'xyzdata')
0042         as = [as varargin{j}];
0043     end
0044     if ischar(varargin{j})
0045         field = varargin{j};
0046     end
0047 end
0048
0049 res = zeros(size(as));
0050 for j=1: numel(as)
0051     try
0052         t = as(j).(field);
0053         res(j) = 1;

```

```
0054     catch
0055         res(j) = 0;
0056     end
0057 end
0058
0059 varargout{1} = res;
0060
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_FTHP.m

win_FTHP

PURPOSE

WIN_FTHP return FTHP window sample for input z.

SYNOPSIS

function w = win_FTHP(z)

DESCRIPTION

WIN_FTHP return FTHP window sample for input z.

M Hewitson 18-05-07

\$Id: win_FTHP.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_FTHP(z)
0002
0003 % WIN_FTHP return FTHP window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_FTHP.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 z = pi * (2 * z - 1);
0011 v = (1 + 1.912510941 ...
0012      * cos (z) + 1.079173272 * cos (2 * z) + ...
0013      0.1832630879 * cos (3 * z));
0014
0015 % Make window struct
0016 w.name      = 'FTHP';
0017 w.alpha     = 0;
0018 w.psll      = 70.4;
0019 w.rov       = 72.3;
0020 w.nenbw     = 3.4279;
0021 w.w3db      = 3.3846;
0022 w.flatness  = 0.0096;
0023 w.ws        = sum(v);
0024 w.ws2       = sum(v.*v);
0025 w.win       = v;
0026
0027 % END
```


[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_submitBtn.m](#)

cb_submitBtn

PURPOSE

SYNOPSIS

```
function cb_submitBtn(varargin)
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function cb_submitBtn(varargin)
0002
0003 myh      = varargin{1};
0004 mainfig = varargin{4};
0005
0006 % get object list
0007 objlist = findobj('Tag', 'workspaceObjsList');
0008 objstr  = get(objlist, 'String');
0009 vals    = get(objlist, 'Value');
0010
0011 % Build sinfo structure
0012 sinfo.conn      = getappdata(mainfig, 'connection');
0013 sinfo.username  = getappdata(mainfig, 'username');
0014 sinfo.experiment_title = get(findobj('Tag', 'experimentTitle'), 'String');
0015 sinfo.experiment_description = get(findobj('Tag', 'experimentDescription'),
'String');
0016 sinfo.reference_ids      = get(findobj('Tag', 'referenceIDs'), 'String');
0017 sinfo.additional_comments = get(findobj('Tag', 'additionalComments'),
'String');
0018 sinfo.additional_authors  = get(findobj('Tag', 'additionalAuthors'),
'String');
0019
0020 if ~isa(sinfo.conn, 'database')
0021     error('### Please connect to a database before trying to submit.');
```

```
0024
0025 % Build structure array
0026 k = 1;
0027 objs = {};
0028 for j=1:length(vals)
0029
0030     objname = deblank(strtok(objstr{vals(j)}));
0031     cmd = sprintf('obj = evalin(''base'', '%s');', objname);
0032     eval(cmd);
0033
0034     for l=1:numel(obj)
0035         objs = [objs {obj(l)}];
0036         k = k + 1;
0037     end
0038 end
0039
0040 % Submit objects
0041 [ids, cid] = ltpda_obj_submit(objs, sinfo);
0042
0043 if isempty(ids)
0044     warning('!!! Failed to submit object: %s', objname);
0045 else
0046     disp(sprintf('++ Submitted object(s) %s (ref id = %s, collection id = %d)',
0047 objname, mat2str(ids), cid));
0047 end
0048
0049
```

[Home](#) > [classes](#) > [@zero](#) > isfield.m

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')
```

VERSION: \$Id: isfield.m,v 1.2 2008/01/02 17:58:51 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = isfield(varargin)
```

```

0002
0003 % ISFIELD tests if the given field is one of the object properties.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: ISFIELD tests if the given field is one of the object
0008 %                properties.
0009 %
0010 % CALL:         isfield(a, 'field');
0011 %
0012 %                The following call returns a parameter list object that
0013 %                contains the default parameter values:
0014 %                >> pl = isfield(zero, 'Params')
0015 %
0016 %                The following call returns the cvs version number:
0017 %                >> ver = isfield(zero, 'Version')
0018 %
0019 % VERSION:       $Id: isfield.m,v 1.2 2008/01/02 17:58:51 ingo Exp $
0020 %
0021 % HISTORY: 25-09-07 M Hewitson
0022 %                Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 VERSION = '$Id: isfield.m,v 1.2 2008/01/02 17:58:51 ingo Exp $';
0027
0028 % 'Params' Call
0029 if nargin == 2
0030     if isa(varargin{1}, 'zero') && strcmp(varargin{2}, 'Params')
0031         varargout{1} = plist();
0032         return
0033     elseif isa(varargin{1}, 'zero') && strcmp(varargin{2}, 'Version')
0034         varargout{1} = VERSION;
0035         return
0036     end
0037 end
0038
0039 as = [];
0040 field = '';
0041 for j=1:nargin
0042     if isa(varargin{j}, 'zero')
0043         as = [as varargin{j}];
0044     end
0045     if ischar(varargin{j})
0046         field = varargin{j};
0047     end
0048 end
0049
0050 res = zeros(size(as));
0051 for j=1:numel(as)
0052     try
0053         t = as(j).(field);

```

```
0054     res(j) = 1;
0055     catch
0056         res(j) = 0;
0057     end
0058 end
0059
0060 varargout{1} = res;
0061
```

[Home](#) > [classes](#) > [@pzmodel](#) > [private](#) > pzm2ab.m

pzm2ab

PURPOSE

PZM2AB convert pzmmodel to IIR filter coefficients using bilinear

SYNOPSIS

function [ao,bo] = pzm2ab(pzm, fs)

DESCRIPTION

PZM2AB convert pzmmodel 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:

- [abccascade](#) 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

```
0001 function [ao,bo] = pzm2ab(pzm, fs)
0002 % PZM2AB convert pzmmodel to IIR filter coefficients using bilinear
0003 % transform.
0004 %
0005 % usage: [a,b] = pzm2ab(pzm, fs)
0006 %
0007 % M Hewitson 03-04-07
```

```

0008 %
0009 % $Id: pzm2ab.m,v 1.4 2007/09/03 09:27:04 hewitson Exp $
0010 %
0011
0012 disp(sprintf('$$$ converting %s', get(pzm, 'name')))
0013
0014 gain = get(pzm, 'gain');
0015 poles = get(pzm, 'poles');
0016 zeros = get(pzm, 'zeros');
0017 np = length(poles);
0018 nz = length(zeros);
0019
0020 ao = [];
0021 bo = [];
0022
0023 % First we should do complex pole/zero pairs
0024 cpoles = [];
0025 for j=1:np
0026     if get(poles(j), 'q') > 0.5
0027         cpoles = [cpoles poles(j)];
0028     end
0029 end
0030 czeros = [];
0031 for j=1:nz
0032     if get(zeros(j), 'q') > 0.5
0033         czeros = [czeros zeros(j)];
0034     end
0035 end
0036
0037 czi = 1;
0038 for j=1:length(cpoles)
0039     if czi <= length(czeros)
0040         % we have a pair
0041         p = cpoles(j);
0042         z = czeros(czi);
0043
0044         [ai,bi] = cpolezero(get(p, 'f'), get(p, 'q'), get(z, 'f'), get(z, 'q'), fs);
0045         if ~isempty(ao)>0
0046             [ao,bo] = abcascade(ao,bo,ai,bi);
0047         else
0048             ao = ai;
0049             bo = bi;
0050         end
0051
0052         % increment zero counter
0053         czi = czi + 1;
0054     end
0055 end
0056
0057 if length(cpoles) > length(czeros)
0058     % do remaining cpoles
0059     for j=length(czeros)+1:length(cpoles)
0060         disp('$$$ computing complex pole');

```

```

0061     [ai,bi] = cp2iir(cpoles(j), fs);
0062     if ~isempty(ao)>0
0063         [ao,bo] = abcascade(ao,bo,ai,bi);
0064     else
0065         ao = ai;
0066         bo = bi;
0067     end
0068 end
0069 else
0070     % do remaining czeros
0071     for j=length(cpoles)+1:length(czeros)
0072         disp('$$$ computing complex zero');
0073         [ai,bi] = cz2iir(czeros(j), fs);
0074         if ~isempty(ao)>0
0075             [ao,bo] = abcascade(ao,bo,ai,bi);
0076         else
0077             ao = ai;
0078             bo = bi;
0079         end
0080     end
0081 end
0082
0083 % Now do the real poles and zeros
0084 for j=1:np
0085     pole = poles(j);
0086     if isnan(get(pole, 'q')) || get(pole, 'q') < 0.5
0087         disp('$$$ computing real pole');
0088         [ai,bi] = rp2iir(pole, fs);
0089         if ~isempty(ao)>0
0090             [ao,bo] = abcascade(ao,bo,ai,bi);
0091         else
0092             ao = ai;
0093             bo = bi;
0094         end
0095     end
0096 end
0097
0098 for j=1:nz
0099     zero = zeros(j);
0100     if isnan(get(zero, 'q')) || get(zero, 'q') < 0.5
0101         disp('$$$ computing real zero');
0102         [ai,bi] = rz2iir(zero, fs);
0103         if ~isempty(ao)>0
0104             [ao,bo] = abcascade(ao,bo,ai,bi);
0105         else
0106             ao = ai;
0107             bo = bi;
0108         end
0109     end
0110 end
0111
0112 % Old version that did everything individually - not so good.

```



```

0113 % for j=1:np
0114 %     pole = poles(j);
0115 %     if get(pole, 'q') > 0.5
0116 %         [ai,bi] = cp2iir(pole, fs);
0117 %     else
0118 %         [ai,bi] = rp2iir(pole, fs);
0119 %     end
0120 %     if ~isempty(ao)>0
0121 %         [ao,bo] = abcascade(ao,bo,ai,bi);
0122 %     else
0123 %         ao = ai;
0124 %         bo = bi;
0125 %     end
0126 % end
0127 %
0128 % for j=1:nz
0129 %     zero = zeros(j);
0130 %     if get(zeros, 'q') > 0.5
0131 %         [ai,bi] = cz2iir(zero, fs);
0132 %     else
0133 %         [ai,bi] = rz2iir(zero, fs);
0134 %     end
0135 %     if ~isempty(ao)>0
0136 %         [ao,bo] = abcascade(ao,bo,ai,bi);
0137 %     else
0138 %         ao = ai;
0139 %         bo = bi;
0140 %     end
0141 % end
0142
0143 ao = ao.*gain;
0144
0145 function [a,b] = cpolezero(pf, pq, zf, zq, fs)
0146 %
0147 % Return IIR filter coefficients for a complex pole
0148 % and complex zero designed using the bilinear transform.
0149 %
0150 % usage: [a,b] = cpolezero(pf, pq, zf, zq, fs)
0151 %
0152 %
0153 % M Hewitson 2003-02-18
0154 %
0155
0156 disp('$$$ computing complex pole/zero pair');
0157
0158 wp = pf*2*pi;
0159 wp2 = wp^2;
0160 wz = zf*2*pi;
0161 wz2 = wz^2;
0162
0163 k = 4*fs*fs + 2*wp*fs/pq + wp2;
0164
0165 a(1) = (4*fs*fs + 2*wz*fs/zq + wz2)/k;

```

```

0166 a(2) = (2*wz2 - 8*fs*fs)/k;
0167 a(3) = (4*fs*fs - 2*wz*fs/zq + wz2)/k;
0168 b(1) = 1;
0169 b(2) = (2*wp2 - 8*fs*fs)/k;
0170 b(3) = (wp2 + 4*fs*fs - 2*wp*fs/pq)/k;
0171
0172 % normalise dc gain to 1
0173 g = iirdcgain(a,b);
0174 a = a / g;
0175
0176
0177 function g = iirdcgain(a,b)
0178
0179 % Work out the DC gain of an IIR
0180 % filter given the coefficients.
0181 %
0182 % usage:
0183 %   g = iirdcgain(a,b)
0184 %
0185 % inputs:
0186 %   a - numerator coefficients
0187 %   b - denominator coefficients
0188 %
0189 % outputs:
0190 %   g - gain
0191 %
0192 % M Hewitson 03-07-02
0193 %
0194 %
0195 % $Id: pzm2ab.m,v 1.4 2007/09/03 09:27:04 hewitson Exp $
0196 %
0197
0198 g = 0;
0199
0200 suma = sum(a);
0201 if(length(b)>1)
0202     sumb = sum(b);
0203     g = suma / sumb;
0204 else
0205     g = suma;
0206 end
0207
0208 % END

```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_FTNI.m

win_FTNI

PURPOSE

WIN_FTNI return FTNI window sample for input z.

SYNOPSIS

function w = win_FTNI(z)

DESCRIPTION

WIN_FTNI return FTNI window sample for input z.

M Hewitson 18-05-07

\$Id: win_FTNI.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_FTNI(z)
0002
0003 % WIN_FTNI return FTNI window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_FTNI.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 z = z * 2 * pi;
0011 v = 0.2810639 - 0.5208972 * cos (z) + 0.1980399 * cos (2 * z);
0012
0013 % Make window struct
0014 w.name      = 'FTNI';
0015 w.alpha     = 0;
0016 w.psll      = 44.4;
0017 w.rov       = 65.6;
0018 w.nenbw     = 2.9656;
0019 w.w3db      = 2.9355;
0020 w.flatness  = 0.0169;
0021 w.ws        = sum(v);
0022 w.ws2       = sum(v.*v);
0023 w.win       = v;
0024
0025 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_tableSelect.m](#)

cb_tableSelect

PURPOSE

SYNOPSIS

```
function cb_tableSelect(varargin)
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

- [buildquery](#)
- [getFields](#) get the selected table

This function is called by:

SOURCE CODE

```
0001 function cb_tableSelect(varargin)
0002
0003 myh      = varargin{1};
0004 mainfig   = findobj('Tag', 'LTPDARepomainfig');
0005 fieldsList = findobj('Tag', 'fieldsList');
0006 Nc        = getappdata(mainfig, 'Nconditions');
0007
0008 % Get conn
0009 conn = getappdata(mainfig, 'connection');
0010
0011 % Set fields in list
0012 fields = getFields(conn);
0013 set(fieldsList, 'Value', 1);
```

```
0014 set(fieldsList, 'String', fields);
0015
0016 % Set fields in condition drop downs
0017 for j=1:Nc
0018     cdbh = findobj('Tag', sprintf('c%02dField', j));
0019     set(cdbh, 'Value', 1);
0020     set(cdbh, 'String', fields);
0021 end
0022
0023 % Build query
0024 buildquery();
0025
0026 setappdata(mainfig, 'tableSelected', get(myh, 'Value'));
0027
```

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

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

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function varargout = islinespec(str)
0002 % ISLINESPEC checks a string to the line spec syntax.
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
```

```

0005 % DESCRIPTION: ISLINESPEC checks a string to the line spec syntax.
0006 %
0007 % CALL:          ret          = islinespec(str);
0008 %               [style_array ret] = islinespec(str);
0009 %
0010 % REMARK:        The style_array is a cell array with the line style, marker style
0011 %               and the color style.
0012 %               style_array{1}: line style
0013 %               style_array{2}: marker style
0014 %               style_array{3}: color style
0015 %
0016 % VERSION:       $Id: islinespec.m,v 1.3 2007/08/24 13:05:09 ingo Exp $
0017 %
0018 % HISTORY:       21-08-2007 Diepholz
0019 %               Creation
0020 %
0021 %%%%%%%%%%%%%%%
0022
0023 if nargin == 0
0024     str = '';
0025 elseif nargin == 1
0026     if ~ischar(str)
0027         error('### The type of the input argument must be a ''char''.');
0028     end
0029 else
0030     error('### Unknown nuber of inputs.');
```

0031 end

```

0032
0033 line_style          = {'-.-', '--', '-', ':'};
0034 full_name_marker_style = {'square', 'diamond', 'pentagram', 'hexagram'};
0035 marker_style =
0036     {'+', 'o', 'O', '*', '.', 'x', '^', ...
0037      '<', '>', 's', 'd', 'h', 'p', };
0037 full_name_color_style = {'red', 'green', 'cyan', 'yellow', ...
0038     'white', 'blue', 'magenta', 'black'};
0039 color_style         = {'r', 'g', 'c', 'y', 'w', 'b', 'm', 'k'};
0040
0041 out_line_style      = '';
0042 out_marker_style    = '';
0043 out_color_style     = '';
0044
0045 copy_str            = str;
0046 found_marker_style  = false;
0047 found_color_style   = false;
0048
0049 %%%%%%%%%%%%%%% Check the line_style %%%%%%%%%%%%%%%
0050 for ii = 1:length(line_style)
0051     idx = strfind(copy_str, line_style{ii});
0052     if ~isempty(idx)
0053         copy_str = strrep(copy_str, line_style{ii}, '');
0054         out_line_style = line_style{ii};
0055         break;
0056     end
0057 end

```



```

0058
0059 %%%%%%%%%% Check full name marker style %%%%%%%%%%
0060 for ii = 1:length(full_name_marker_style)
0061     idx = strfind(copy_str, full_name_marker_style{ii});
0062     if ~isempty(idx)
0063         copy_str = strrep(copy_str, full_name_marker_style{ii}, '');
0064         out_marker_style = full_name_marker_style{ii};
0065         found_marker_style = true;
0066         break;
0067     end
0068 end
0069
0070 %%%%%%%%%% Check full name color style %%%%%%%%%%
0071 for ii = 1:length(full_name_color_style)
0072     idx = strfind(copy_str, full_name_color_style{ii});
0073     if ~isempty(idx)
0074         copy_str = strrep(copy_str, full_name_color_style{ii}, '');
0075         out_color_style = full_name_color_style{ii};
0076         found_color_style = true;
0077         break;
0078     end
0079 end
0080
0081 %%%%%%%%%% Check full marker style %%%%%%%%%%
0082 if found_marker_style == false
0083     for ii = 1:length(marker_style)
0084         idx = strfind(copy_str, marker_style{ii});
0085         if ~isempty(idx)
0086             copy_str = strrep(copy_str, marker_style{ii}, '');
0087             out_marker_style = marker_style{ii};
0088             break;
0089         end
0090     end
0091 end
0092
0093 %%%%%%%%%% Check full color style %%%%%%%%%%
0094 if found_color_style == false
0095     for ii = 1:length(color_style)
0096         idx = strfind(copy_str, color_style{ii});
0097         if ~isempty(idx)
0098             copy_str = strrep(copy_str, color_style{ii}, '');
0099             out_color_style = color_style{ii};
0100             break;
0101         end
0102     end
0103 end
0104
0105 %%%%%%%%%% Set the output %%%%%%%%%%
0106 if isempty(copy_str)
0107     ret = true;
0108 else
0109     ret = false;
0110 end

```

```
0111
0112     if nargout == 0 || nargout == 1
0113         varargout{1} = ret;
0114     elseif nargout == 2
0115         varargout{1} = ret;
0116         varargout{2} = {out_line_style, out_marker_style, out_color_style};
0117     else
0118         error('### Unknown numbers of outputs.');
```

```
0119     end
0120
0121 end
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_FTSRS.m

win_FTSRS

PURPOSE [^](#)

WIN_FTSRS return FTSRS window sample for input z.

SYNOPSIS [^](#)

```
function w = win_FTSRS(z)
```

DESCRIPTION [^](#)

WIN_FTSRS return FTSRS window sample for input z.

M Hewitson 18-05-07

\$Id: win_FTSRS.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^](#)

This function calls:

This function is called by:

SOURCE CODE [^](#)

```
0001 function w = win_FTSRS(z)
0002
0003 % WIN_FTSRS return FTSRS window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_FTSRS.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 z = z * 2 * pi;
0011 v = (1.0 - 1.93 * cos (z) + 1.29 ...
0012      * cos (2 * z) - 0.388 * cos (3 * z) + 0.028 * cos (4 * z));
0013
0014 % Make window struct
0015 w.name      = 'FTSRS';
0016 w.alpha     = 0;
0017 w.psll      = 76.6;
0018 w.rov       = 75.4;
0019 w.nenbw     = 3.7702;
0020 w.w3db      = 3.7274;
0021 w.flatness  = -0.0156;
0022 w.ws        = sum(v);
0023 w.ws2       = sum(v.*v);
0024 w.win       = v;
0025
0026 % END
```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > cb_workspaceObjsList.m

cb_workspaceObjsList

PURPOSE

SYNOPSIS

```
function cb_workspaceObjsList(varargin)
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function cb_workspaceObjsList(varargin)
0002
0003 myh      = varargin{1};
0004 mainfig = varargin{4};
0005
0006 % get the selected line
0007 val = get(myh, 'Value');
0008 if length(val) == 1
0009     lines = get(myh, 'String');
0010     objname = deblank(strtok(lines{val}));
0011
0012     cmd = sprintf('obj = evalin(''base'', '%s'');', objname);
0013     eval(cmd);
0014
0015     objtxt = display(obj);
0016 else
0017     objtxt = '';
```

```
0018 end
0019
0020 % set objDisplay Txt
0021 ltpdaObjInfo = findobj('Tag', 'ltpdaObjInfo');
0022 set(ltpdaObjInfo, 'String', objtxt)
0023
0024
0025
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [pzmodel_designer](#) > pzmodel_helper.m

pzmodel_helper

PURPOSE

PZMODEL_HELPER M-file for pzmodel_helper.fig

SYNOPSIS

function varargout = pzmodel_helper(varargin)

DESCRIPTION

PZMODEL_HELPER M-file for pzmodel_helper.fig
PZMODEL_HELPER, by itself, creates a new PZMODEL_HELPER or raises the existing singleton*.

H = PZMODEL_HELPER returns the handle to a new PZMODEL_HELPER or the handle to the existing singleton*.

PZMODEL_HELPER('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PZMODEL_HELPER.M with the given input arguments.

PZMODEL_HELPER('Property','Value',...) creates a new PZMODEL_HELPER or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before pzmodel_helper_OpeningFunction gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to pzmodel_helper_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SUBFUNCTIONS

- [function pzmodel_helper_OpeningFcn\(hObject, eventdata, handles, varargin\)](#)
- [function varargout = pzmodel_helper_OutputFcn\(hObject, eventdata, handles\)](#)
- [function pzEdit_Callback\(hObject, eventdata, handles\)](#)

- function pzEdit_CreateFcn(hObject, eventdata, handles)
- function addPoleBtn_Callback(hObject, eventdata, handles)
- function poleList_Callback(hObject, eventdata, handles)
- function poleList_CreateFcn(hObject, eventdata, handles)
- function addZeroBtn_Callback(hObject, eventdata, handles)
- function zeroList_Callback(hObject, eventdata, handles)
- function zeroList_CreateFcn(hObject, eventdata, handles)
- function updatePolesList(handles)
- function updateZerosList(handles)
- function updateModelResp(handles)
- function gainEdit_Callback(hObject, eventdata, handles)
- function gainEdit_CreateFcn(hObject, eventdata, handles)
- function f1Edit_Callback(hObject, eventdata, handles)
- function f1Edit_CreateFcn(hObject, eventdata, handles)
- function f2Edit_Callback(hObject, eventdata, handles)
- function f2Edit_CreateFcn(hObject, eventdata, handles)
- function nfEdit_Callback(hObject, eventdata, handles)
- function nfEdit_CreateFcn(hObject, eventdata, handles)
- function replotBtn_Callback(hObject, eventdata, handles)
- function fsEdit_Callback(hObject, eventdata, handles)
- function fsEdit_CreateFcn(hObject, eventdata, handles)
- function figureBtn_Callback(hObject, eventdata, handles)
- function filterStrEdit_Callback(hObject, eventdata, handles)
- function filterStrEdit_CreateFcn(hObject, eventdata, handles)
- function clearBtn_Callback(hObject, eventdata, handles)
- function deletePolesBtn_Callback(hObject, eventdata, handles)
- function deleteZerosBtn_Callback(hObject, eventdata, handles)
- function saveAObtn_Callback(hObject, eventdata, handles)

SOURCE CODE

```

0001 function varargout = pzmodel_helper(varargin)
0002 % PZMODEL_HELPER M-file for pzmodel_helper.fig
0003 %     PZMODEL_HELPER, by itself, creates a new PZMODEL_HELPER or raises the
existing
0004 %     singleton*.
0005 %
0006 %     H = PZMODEL_HELPER returns the handle to a new PZMODEL_HELPER or the
handle to
0007 %     the existing singleton*.
0008 %
0009 %     PZMODEL_HELPER('CALLBACK',hObject,eventData,handles,...) calls the local
0010 %     function named CALLBACK in PZMODEL_HELPER.M with the given input
arguments.
0011 %
0012 %     PZMODEL_HELPER('Property','Value',...) creates a new PZMODEL_HELPER or
raises the
0013 %     existing singleton*. Starting from the left, property value pairs are
0014 %     applied to the GUI before pzmodel_helper_OpeningFunction gets called. An

```



```

0015 % unrecognized property name or invalid value makes property application
0016 % stop. All inputs are passed to pzmodel_helper_OpeningFcn via varargin.
0017 %
0018 % *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
0019 % instance to run (singleton)".
0020 %
0021 % See also: GUIDE, GUIDATA, GUIHANDLES
0022
0023 % Edit the above text to modify the response to help pzmodel_helper
0024
0025 % Last Modified by GUIDE v2.5 19-Apr-2007 10:37:13
0026
0027 % Begin initialization code - DO NOT EDIT
0028 gui_Singleton = 1;
0029 gui_State = struct('gui_Name', mfilename, ...
0030 'gui_Singleton', gui_Singleton, ...
0031 'gui_OpeningFcn', @pzmodel_helper_OpeningFcn, ...
0032 'gui_OutputFcn', @pzmodel_helper_OutputFcn, ...
0033 'gui_LayoutFcn', [] , ...
0034 'gui_Callback', []);
0035 if nargin && ischar(varargin{1})
0036     gui_State.gui_Callback = str2func(varargin{1});
0037 end
0038
0039 if nargout
0040     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
0041 else
0042     gui_mainfcn(gui_State, varargin{:});
0043 end
0044 % End initialization code - DO NOT EDIT
0045
0046
0047 % --- Executes just before pzmodel_helper is made visible.
0048 function pzmodel_helper_OpeningFcn(hObject, eventdata, handles, varargin)
0049 % This function has no output args, see OutputFcn.
0050 % hObject handle to figure
0051 % eventdata reserved - to be defined in a future version of MATLAB
0052 % handles structure with handles and user data (see GUIDATA)
0053 % varargin command line arguments to pzmodel_helper (see VARARGIN)
0054
0055 % Choose default command line output for pzmodel_helper
0056 handles.output = hObject;
0057
0058 % Update handles structure
0059 guidata(hObject, handles);
0060
0061 % UIWAIT makes pzmodel_helper wait for user response (see UIRESUME)
0062 % uiwait(handles.main);
0063
0064 % create empty lists
0065 poles = [];
0066 zeros = [];

```

```

0067
0068 setappdata(handles.main, 'poles', poles);
0069 setappdata(handles.main, 'zeros', zeros);
0070 setappdata(handles.main, 'filt', []);
0071
0072 % --- Outputs from this function are returned to the command line.
0073 function varargout = pzmodel_helper_OutputFcn(hObject, eventdata, handles)
0074 % varargout    cell array for returning output args (see VARARGOUT);
0075 % hObject      handle to figure
0076 % eventdata    reserved - to be defined in a future version of MATLAB
0077 % handles      structure with handles and user data (see GUIDATA)
0078
0079 % Get default command line output from handles structure
0080 varargout{1} = handles.output;
0081
0082
0083
0084 function pzEdit_Callback(hObject, eventdata, handles)
0085 % hObject      handle to pzEdit (see GCBO)
0086 % eventdata    reserved - to be defined in a future version of MATLAB
0087 % handles      structure with handles and user data (see GUIDATA)
0088
0089 % Hints: get(hObject,'String') returns contents of pzEdit as text
0090 %          str2double(get(hObject,'String')) returns contents of pzEdit as a
double
0091
0092
0093 % --- Executes during object creation, after setting all properties.
0094 function pzEdit_CreateFcn(hObject, eventdata, handles)
0095 % hObject      handle to pzEdit (see GCBO)
0096 % eventdata    reserved - to be defined in a future version of MATLAB
0097 % handles      empty - handles not created until after all CreateFcns called
0098
0099 % Hint: edit controls usually have a white background on Windows.
0100 %          See ISPC and COMPUTER.
0101 if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
0102     set(hObject,'BackgroundColor','white');
0103 end
0104
0105
0106 % --- Executes on button press in addPoleBtn.
0107 function addPoleBtn_Callback(hObject, eventdata, handles)
0108 % hObject      handle to addPoleBtn (see GCBO)
0109 % eventdata    reserved - to be defined in a future version of MATLAB
0110 % handles      structure with handles and user data (see GUIDATA)
0111
0112 % get values from pzEdit
0113 strVal = get(handles.pzEdit, 'String');
0114 val    = str2num(strVal);
0115 f      = val(1);
0116 if length(val) == 2

```

```

0117     q = val(2);
0118     p = pole(f,q);
0119 else
0120     p = pole(f);
0121 end
0122
0123 poles = getappdata(handles.main, 'poles');
0124 poles = [poles p];
0125 setappdata(handles.main, 'poles', poles);
0126
0127 % update list
0128 updatePolesList(handles);
0129
0130 % update model
0131 updateModelResp(handles);
0132
0133
0134 % --- Executes on selection change in poleList.
0135 function poleList_Callback(hObject, eventdata, handles)
0136 % hObject      handle to poleList (see GCBO)
0137 % eventdata    reserved - to be defined in a future version of MATLAB
0138 % handles      structure with handles and user data (see GUIDATA)
0139
0140 % Hints: contents = get(hObject,'String') returns poleList contents as cell
array
0141 %           contents{get(hObject,'Value')} returns selected item from poleList
0142
0143
0144 % --- Executes during object creation, after setting all properties.
0145 function poleList_CreateFcn(hObject, eventdata, handles)
0146 % hObject      handle to poleList (see GCBO)
0147 % eventdata    reserved - to be defined in a future version of MATLAB
0148 % handles      empty - handles not created until after all CreateFcns called
0149
0150 % Hint: listbox controls usually have a white background on Windows.
0151 %           See ISPC and COMPUTER.
0152 if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
0153     set(hObject,'BackgroundColor','white');
0154 end
0155
0156
0157 % --- Executes on button press in addZeroBtn.
0158 function addZeroBtn_Callback(hObject, eventdata, handles)
0159 % hObject      handle to addZeroBtn (see GCBO)
0160 % eventdata    reserved - to be defined in a future version of MATLAB
0161 % handles      structure with handles and user data (see GUIDATA)
0162
0163 % get values from pzEdit
0164 strVal = get(handles.pzEdit, 'String');
0165 val    = str2num(strVal);
0166 f      = val(1);

```

```

0167 if length(val) == 2
0168     q = val(2);
0169     z = zero(f,q);
0170 else
0171     z = zero(f);
0172 end
0173
0174 zeros = getappdata(handles.main, 'zeros');
0175 zeros = [zeros z];
0176 setappdata(handles.main, 'zeros', zeros);
0177
0178 % update list
0179 updateZerosList(handles);
0180
0181 % update model
0182 updateModelResp(handles);
0183
0184
0185 % --- Executes on selection change in zeroList.
0186 function zeroList_Callback(hObject, eventdata, handles)
0187 % hObject      handle to zeroList (see GCBO)
0188 % eventdata    reserved - to be defined in a future version of MATLAB
0189 % handles      structure with handles and user data (see GUIDATA)
0190
0191 % Hints: contents = get(hObject,'String') returns zeroList contents as cell
array
0192 %           contents{get(hObject,'Value')} returns selected item from zeroList
0193
0194
0195 % --- Executes during object creation, after setting all properties.
0196 function zeroList_CreateFcn(hObject, eventdata, handles)
0197 % hObject      handle to zeroList (see GCBO)
0198 % eventdata    reserved - to be defined in a future version of MATLAB
0199 % handles      empty - handles not created until after all CreateFcns called
0200
0201 % Hint: listbox controls usually have a white background on Windows.
0202 %           See ISPC and COMPUTER.
0203 if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUiControlBackgroundColor'))
0204     set(hObject,'BackgroundColor','white');
0205 end
0206
0207 %----- Update Poles List -----
0208 function updatePolesList(handles)
0209
0210 poles = getappdata(handles.main, 'poles');
0211 pstr = [];
0212 for n=1:length(poles)
0213     p = poles(n);
0214     f = get(p, 'f');
0215     q = get(p, 'q');
0216     if q>0.5

```

```

0217     pstr = strvcat(pstr, sprintf('%2.2f Hz   Q=%2.2f', f, q));
0218 else
0219     pstr = strvcat(pstr, sprintf('%2.2f Hz', f));
0220 end
0221 end
0222
0223 if isempty(poles)
0224     set(handles.poleList, 'String', ' ');
0225 else
0226     set(handles.poleList, 'String', pstr);
0227 end
0228 set(handles.poleList, 'Value', 1);
0229
0230 %----- Update Zeros List -----
0231 function updateZerosList(handles)
0232
0233 zeros = getappdata(handles.main, 'zeros');
0234 pstr = [];
0235 for n=1:length(zeros)
0236     p = zeros(n);
0237     f = get(p, 'f');
0238     q = get(p, 'q');
0239     if q>0.5
0240         pstr = strvcat(pstr, sprintf('%2.2f Hz   Q=%2.2f', f, q));
0241     else
0242         pstr = strvcat(pstr, sprintf('%2.2f Hz', f));
0243     end
0244 end
0245
0246 if isempty(zeros)
0247     set(handles.zeroList, 'String', ' ');
0248 else
0249     set(handles.zeroList, 'String', pstr);
0250 end
0251 set(handles.zeroList, 'Value', 1);
0252
0253 %----- Update Model response -----
0254 function updateModelResp(handles)
0255
0256 gain = str2double(get(handles.gainEdit, 'String'));
0257 poles = getappdata(handles.main, 'poles');
0258 zeros = getappdata(handles.main, 'zeros');
0259
0260 pzm = pzmodel(gain, poles, zeros);
0261
0262 f1s = get(handles.f1Edit, 'String');
0263 f2s = get(handles.f2Edit, 'String');
0264 nfs = get(handles.nfEdit, 'String');
0265 fss = get(handles.fsEdit, 'String');
0266
0267 if isempty(f1s)
0268     f1 = getlowerFreq(pzm)/10;
0269 else

```

```

0270     f1 = str2double(f1s);
0271 end
0272 if isempty(f2s)
0273     f2 = getupperFreq(pzm)*10;
0274 else
0275     f2 = str2double(f2s);
0276 end
0277 if isempty(nfs)
0278     nf = 1000;
0279 else
0280     nf = str2double(nfs);
0281 end
0282
0283 % make model response
0284 a = resp(pzm, f1, f2, nf);
0285 d = a.data;
0286 mx = d.x;
0287 my = d.y;
0288
0289 % make IIR response
0290 if isempty(fss)
0291     filt = miir(plist([param('pzmodel', pzm)]));
0292 else
0293     filt = miir(plist([param('pzmodel', pzm) param('fs', str2double(fss))]));
0294 end
0295 setappdata(handles.main, 'filt', filt);
0296 set(handles.xilterStrEdit, 'String', string(pzm));
0297
0298 filtresp = resp(filt, plist(param('f', mx.')));
0299 d = filtresp.data;
0300 fx = d.x;
0301 fy = d.y;
0302
0303 axes(handles.magAxes) % Select the proper axes
0304 loglog(mx,abs(my), fx, abs(fy), 'r--')
0305 grid on;
0306 axis tight
0307 xlabel('');
0308 ylabel('Magnitude');
0309 legend('Pole/Zero model', 'IIR', 'Location', 'SouthWest')
0310
0311 axes(handles.phaseAxes) % Select the proper axes
0312 semilogx(mx, ltpda_phase(my), fx, ltpda_phase(fy), 'r--')
0313 grid on;
0314 axis tight
0315 xlabel('Frequency [Hz]');
0316 ylabel('Phase [deg]');
0317
0318
0319
0320
0321
0322 function gainEdit_Callback(hObject, eventdata, handles)

```

```

0323 % hObject      handle to gainEdit (see GCBO)
0324 % eventdata    reserved - to be defined in a future version of MATLAB
0325 % handles      structure with handles and user data (see GUIDATA)
0326
0327 % Hints: get(hObject,'String') returns contents of gainEdit as text
0328 %             str2double(get(hObject,'String')) returns contents of gainEdit as a
double
0329
0330 updateModelResp(handles)
0331
0332 % --- Executes during object creation, after setting all properties.
0333 function gainEdit_CreateFcn(hObject, eventdata, handles)
0334 % hObject      handle to gainEdit (see GCBO)
0335 % eventdata    reserved - to be defined in a future version of MATLAB
0336 % handles      empty - handles not created until after all CreateFcns called
0337
0338 % Hint: edit controls usually have a white background on Windows.
0339 %             See ISPC and COMPUTER.
0340 if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
0341     set(hObject,'BackgroundColor','white');
0342 end
0343
0344
0345
0346 function f1Edit_Callback(hObject, eventdata, handles)
0347 % hObject      handle to f1Edit (see GCBO)
0348 % eventdata    reserved - to be defined in a future version of MATLAB
0349 % handles      structure with handles and user data (see GUIDATA)
0350
0351 % Hints: get(hObject,'String') returns contents of f1Edit as text
0352 %             str2double(get(hObject,'String')) returns contents of f1Edit as a
double
0353
0354
0355 % --- Executes during object creation, after setting all properties.
0356 function f1Edit_CreateFcn(hObject, eventdata, handles)
0357 % hObject      handle to f1Edit (see GCBO)
0358 % eventdata    reserved - to be defined in a future version of MATLAB
0359 % handles      empty - handles not created until after all CreateFcns called
0360
0361 % Hint: edit controls usually have a white background on Windows.
0362 %             See ISPC and COMPUTER.
0363 if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
0364     set(hObject,'BackgroundColor','white');
0365 end
0366
0367
0368
0369 function f2Edit_Callback(hObject, eventdata, handles)
0370 % hObject      handle to f2Edit (see GCBO)

```



```

0371 % eventdata reserved - to be defined in a future version of MATLAB
0372 % handles structure with handles and user data (see GUIDATA)
0373
0374 % Hints: get(hObject,'String') returns contents of f2Edit as text
0375 % str2double(get(hObject,'String')) returns contents of f2Edit as a
double
0376
0377
0378 % --- Executes during object creation, after setting all properties.
0379 function f2Edit_CreateFcn(hObject, eventdata, handles)
0380 % hObject handle to f2Edit (see GCBO)
0381 % eventdata reserved - to be defined in a future version of MATLAB
0382 % handles empty - handles not created until after all CreateFcns called
0383
0384 % Hint: edit controls usually have a white background on Windows.
0385 % See ISPC and COMPUTER.
0386 if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUiControlBackgroundColor'))
0387 set(hObject,'BackgroundColor','white');
0388 end
0389
0390
0391
0392 function nfEdit_Callback(hObject, eventdata, handles)
0393 % hObject handle to nfEdit (see GCBO)
0394 % eventdata reserved - to be defined in a future version of MATLAB
0395 % handles structure with handles and user data (see GUIDATA)
0396
0397 % Hints: get(hObject,'String') returns contents of nfEdit as text
0398 % str2double(get(hObject,'String')) returns contents of nfEdit as a
double
0399
0400
0401 % --- Executes during object creation, after setting all properties.
0402 function nfEdit_CreateFcn(hObject, eventdata, handles)
0403 % hObject handle to nfEdit (see GCBO)
0404 % eventdata reserved - to be defined in a future version of MATLAB
0405 % handles empty - handles not created until after all CreateFcns called
0406
0407 % Hint: edit controls usually have a white background on Windows.
0408 % See ISPC and COMPUTER.
0409 if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUiControlBackgroundColor'))
0410 set(hObject,'BackgroundColor','white');
0411 end
0412
0413
0414 % --- Executes on button press in replotBtn.
0415 function replotBtn_Callback(hObject, eventdata, handles)
0416 % hObject handle to replotBtn (see GCBO)
0417 % eventdata reserved - to be defined in a future version of MATLAB
0418 % handles structure with handles and user data (see GUIDATA)

```



```

0419
0420 updateModelResp(handles)
0421
0422
0423
0424 function fsEdit_Callback(hObject, eventdata, handles)
0425 % hObject      handle to fsEdit (see GCBO)
0426 % eventdata    reserved - to be defined in a future version of MATLAB
0427 % handles      structure with handles and user data (see GUIDATA)
0428
0429 % Hints: get(hObject,'String') returns contents of fsEdit as text
0430 %           str2double(get(hObject,'String')) returns contents of fsEdit as a
double
0431 updateModelResp(handles)
0432
0433
0434 % --- Executes during object creation, after setting all properties.
0435 function fsEdit_CreateFcn(hObject, eventdata, handles)
0436 % hObject      handle to fsEdit (see GCBO)
0437 % eventdata    reserved - to be defined in a future version of MATLAB
0438 % handles      empty - handles not created until after all CreateFcns called
0439
0440 % Hint: edit controls usually have a white background on Windows.
0441 %           See ISPC and COMPUTER.
0442 if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
0443     set(hObject,'BackgroundColor','white');
0444 end
0445
0446
0447 % --- Executes on button press in figureBtn.
0448 function figureBtn_Callback(hObject, eventdata, handles)
0449 % hObject      handle to figureBtn (see GCBO)
0450 % eventdata    reserved - to be defined in a future version of MATLAB
0451 % handles      structure with handles and user data (see GUIDATA)
0452
0453 gain = str2double(get(handles.gainEdit, 'String'));
0454 poles = getappdata(handles.main, 'poles');
0455 zeros = getappdata(handles.main, 'zeros');
0456
0457 pzm = pzmodel(gain, poles, zeros);
0458
0459 f1s = get(handles.f1Edit, 'String');
0460 f2s = get(handles.f2Edit, 'String');
0461 nfs = get(handles.nfEdit, 'String');
0462 fss = get(handles.fsEdit, 'String');
0463
0464 if isempty(f1s)
0465     f1 = getlowerFreq(pzm)/10;
0466 else
0467     f1 = str2double(f1s);
0468 end

```

```

0469 if isempty(f2s)
0470     f2 = getupperFreq(pzm)*10;
0471 else
0472     f2 = str2double(f2s);
0473 end
0474 if isempty(nfs)
0475     nf = 1000;
0476 else
0477     nf = str2double(nfs);
0478 end
0479
0480 % make model response
0481 a = resp(pzm, f1, f2, nf);
0482 d = a.data;
0483 mx = d.x;
0484 my = d.y;
0485
0486 % make IIR response
0487 if isempty(fss)
0488     filt = miir(plist([param('pzmodel', pzm)]));
0489 else
0490     filt = miir(plist([param('pzmodel', pzm) param('fs', str2double(fss))]));
0491 end
0492
0493
0494 filtresp = resp(filt, plist(param('f', mx.')));
0495 d = filtresp.data;
0496 fx = d.x;
0497 fy = d.y;
0498
0499 figure
0500 subplot(3,1,1:2)
0501 loglog(mx,abs(my), fx, abs(fy), 'r--')
0502 grid on;
0503 axis tight
0504 xlabel('');
0505 ylabel('Magnitude');
0506 legend('Pole/Zero model', 'IIR', 'Location', 'SouthWest')
0507
0508 subplot(3,1,3)
0509 semilogx(mx, ltpda_phase(my), fx, ltpda_phase(fy), 'r--')
0510 grid on;
0511 axis tight
0512 xlabel('Frequency [Hz]');
0513 ylabel('Phase [deg]');
0514
0515
0516 function filterStrEdit_Callback(hObject, eventdata, handles)
0517 % hObject    handle to filterStrEdit (see GCBO)
0518 % eventdata  reserved - to be defined in a future version of MATLAB
0519 % handles    structure with handles and user data (see GUIDATA)
0520
0521 % Hints: get(hObject,'String') returns contents of filterStrEdit as text

```

```

0522 %           str2double(get(hObject,'String')) returns contents of filterStrEdit as
a double
0523
0524
0525 % --- Executes during object creation, after setting all properties.
0526 function filterStrEdit_CreateFcn(hObject, eventdata, handles)
0527 % hObject     handle to filterStrEdit (see GCBO)
0528 % eventdata   reserved - to be defined in a future version of MATLAB
0529 % handles     empty - handles not created until after all CreateFcns called
0530
0531 % Hint: edit controls usually have a white background on Windows.
0532 %           See ISPC and COMPUTER.
0533 if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUiControlBackgroundColor'))
0534     set(hObject,'BackgroundColor','white');
0535 end
0536
0537
0538 % --- Executes on button press in clearBtn.
0539 function clearBtn_Callback(hObject, eventdata, handles)
0540 % hObject     handle to clearBtn (see GCBO)
0541 % eventdata   reserved - to be defined in a future version of MATLAB
0542 % handles     structure with handles and user data (see GUIDATA)
0543
0544 setappdata(handles.main, 'poles', []);
0545 setappdata(handles.main, 'zeros', []);
0546
0547 % update list
0548 updateZerosList(handles);
0549 updatePolesList(handles);
0550
0551 % update model
0552 updateModelResp(handles);
0553
0554
0555 % --- Executes on button press in deletePolesBtn.
0556 function deletePolesBtn_Callback(hObject, eventdata, handles)
0557 % hObject     handle to deletePolesBtn (see GCBO)
0558 % eventdata   reserved - to be defined in a future version of MATLAB
0559 % handles     structure with handles and user data (see GUIDATA)
0560
0561 % get selection from list
0562 values = get(handles.poleList, 'Value');
0563
0564 % get poles
0565 poles = getappdata(handles.main, 'poles');
0566
0567 po = [];
0568 for j=1:length(poles)
0569     if j~=values
0570         po = [po poles(j)];
0571     end

```

```

0572 end
0573
0574 setappdata(handles.main, 'poles', po);
0575 updatePolesList(handles);
0576
0577 % update model
0578 updateModelResp(handles);
0579
0580
0581 % --- Executes on button press in deleteZerosBtn.
0582 function deleteZerosBtn_Callback(hObject, eventdata, handles)
0583 % hObject      handle to deleteZerosBtn (see GCBO)
0584 % eventdata    reserved - to be defined in a future version of MATLAB
0585 % handles      structure with handles and user data (see GUIDATA)
0586
0587 % get selection from list
0588 values = get(handles.zeroList, 'Value');
0589 % get zeros
0590 zeros = getappdata(handles.main, 'zeros');
0591 zo = [];
0592 for j=1:length(zeros)
0593     if j~=values
0594         zo = [zo zeros(j)];
0595     end
0596 end
0597
0598 setappdata(handles.main, 'zeros', zo);
0599 updateZerosList(handles);
0600
0601 % update model
0602 updateModelResp(handles);
0603
0604
0605 % --- Executes on button press in saveA0btn.
0606 function saveA0btn_Callback(hObject, eventdata, handles)
0607 % hObject      handle to saveA0btn (see GCBO)
0608 % eventdata    reserved - to be defined in a future version of MATLAB
0609 % handles      structure with handles and user data (see GUIDATA)
0610
0611 % get response
0612 gain = str2double(get(handles.gainEdit, 'String'));
0613 poles = getappdata(handles.main, 'poles');
0614 zeros = getappdata(handles.main, 'zeros');
0615 pzm = pzmodel(gain, poles, zeros);
0616
0617 f1s = get(handles.f1Edit, 'String');
0618 f2s = get(handles.f2Edit, 'String');
0619 nfs = get(handles.nfEdit, 'String');
0620 fss = get(handles.fsEdit, 'String');
0621
0622 if isempty(f1s)
0623     f1 = getLowerFreq(pzm)/10;

```

```
0624 else
0625     f1 = str2double(f1s);
0626 end
0627 if isempty(f2s)
0628     f2 = getupperFreq(pzm)*10;
0629 else
0630     f2 = str2double(f2s);
0631 end
0632 if isempty(nfs)
0633     nf = 1000;
0634 else
0635     nf = str2double(nfs);
0636 end
0637
0638 % make model response
0639 a = resp(pzm, f1, f2, nf);
0640
0641 % Get filename
0642 [filename, pathname] = uiputfile('*.xml', 'LTPDA XML file (*.xml)', 'Save as');
0643 if isequal(filename,0) || isequal(pathname,0)
0644 else
0645     fname = fullfile(pathname, filename);
0646     save(a, fname);
0647 end
0648
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_HFT116D.m

win_HFT116D

PURPOSE

WIN_HFT116D return HFT116D window sample for input z.

SYNOPSIS

```
function w = win_HFT116D(z)
```

DESCRIPTION

WIN_HFT116D return HFT116D window sample for input z.

M Hewitson 18-05-07

\$Id: win_HFT116D.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_HFT116D(z)
0002
0003 % WIN_HFT116D return HFT116D window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT116D.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 z = z * 2 * pi;
0011 v = (1 - 1.9575375 * cos (z) + ...
0012      1.4780705 * cos (2 * z) - 0.6367431 * cos (3 * z) + ...
0013      0.1228389 * cos (4 * z) - 0.0066288 * cos (5 * z));
0014
0015 % Make window struct
0016 w.name      = 'HFT116D';
0017 w.alpha     = 0;
0018 w.psll      = 116.8;
0019 w.rov       = 78.2;
0020 w.nenbw     = 4.2186;
0021 w.w3db      = 4.1579;
0022 w.flatness  = -0.0028;
0023 w.ws        = sum(v);
0024 w.ws2       = sum(v.*v);
0025 w.win       = v;
0026
0027 % END
```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > cb_workspaceObjsListQL.m

cb_workspaceObjsListQL

PURPOSE

SYNOPSIS

```
function cb_workspaceObjsListQL(varargin)
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

- [fillClassMethodsList](#) Get the object types

This function is called by:

SOURCE CODE

```
0001 function cb_workspaceObjsListQL(varargin)
0002
0003 myh      = varargin{1};
0004 mainfig = varargin{4};
0005
0006 % get the selected line
0007 val = get(myh, 'Value');
0008 if length(val) == 1
0009     lines = get(myh, 'String');
0010     objname = deblank(strtok(lines{val}));
0011
0012     cmd = sprintf('obj = evalin(''base'', '%s');', objname);
0013     eval(cmd);
0014
```



```
0015     objtxt = display(obj);
0016 else
0017     objtxt = '';
0018 end
0019
0020 % set objDisplay Txt
0021 ltpdaObjInfo = findobj('Tag', 'ltpdaObjInfoQL');
0022 set(ltpdaObjInfo, 'String', objtxt)
0023
0024 fillClassMethodsList();
0025
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_HFT144D.m

win_HFT144D

PURPOSE

WIN_HFT144D return HFT144D window sample for input z.

SYNOPSIS

```
function w = win_HFT144D(z)
```

DESCRIPTION

```
WIN_HFT144D return HFT144D window sample for input z.
```

```
M Hewitson 18-05-07
```

```
$Id: win_HFT144D.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
```

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_HFT144D(z)
0002
0003 % WIN_HFT144D return HFT144D window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT144D.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 z = z * 2 * pi;
0011 v = (1 - 1.96760033 * cos (z) ...
0012      + 1.57983607 * cos (2 * z) - 0.81123644 * cos (3 * z) + ...
0013      0.22583558 * cos (4 * z) - 0.02773848 * cos (5 * z) + ...
0014      0.00090360 * cos (6 * z));
0015
0016 % Make window struct
0017 w.name      = 'HFT144D';
0018 w.alpha     = 0;
0019 w.psll      = 114.1;
0020 w.rov       = 79.9;
0021 w.nenbw     = 4.5386;
0022 w.w3db     = 4.4697;
0023 w.flatness  = 0.0021;
0024 w.ws       = sum(v);
0025 w.ws2      = sum(v.*v);
0026 w.win       = v;
0027
0028 % END
```

[Home](#) > [classes](#) > [@cdata](#) > cdata.m

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:

- **cdata** CDATA constant data object class constructor.
- **set** SET set a cdata property.
- **xmlparse** XMLPARSE creates a cdata from a cdata xml file.

This function is called by:

- **cdata** CDATA constant data object class constructor.

SUBFUNCTIONS

- **function c = init()**
- **function c = checkCdata(c)**

SOURCE CODE

```

0001 function c = cdata(varargin)
0002 % CDATA constant data object class constructor.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CDATA constant data object class constructor.
0007 %             Create a constant data object.
0008 %
0009 %     Properties:
0010 %         name      - name of data object
0011 %         y          - data values
0012 %         x          - cell array: could be one descriptive tag per value
0013 %         xunits     - this is used to label the x-axis on plots.
0014 %         yunits     - this is used to label the y-axis on plots.
0015 %         version    - version of the constructor code
0016 %         created    - creation time of this fsdata object.
0017 %
0018 %     Possible constructors:
0019 %         c = cdata()      - creates a blank data object
0020 %         c = cdata(pl)   - creates a data object with the given
0021 %                           parameter list which contains 'N' and 'Val'
0022 %         c = cdata(vals) - creates a data object with the given
0023 %                           data values.
0024 %         c = cdata(N,v)  - creates a data object with N samples all of
0025 %                           value, v.
0026 %
0027 % HISTORY: 30-01-2007 Hewitson
0028 %         Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION    = '$Id: cdata.m,v 1.16 2007/11/30 19:36:04 ingo Exp $';
0033
0034 %%%%%%    Is this a 'Params' call?    %%%%%%

```

```

0035 if nargin == 2 && isa(varargin{1}, 'cdata') && ischar(varargin{2})
0036     if strcmp(varargin{2}, 'Params')
0037         c = plist();
0038         return
0039     elseif strcmp(varargin{2}, 'Version')
0040         c = VERSION;
0041         return
0042     end
0043 end
0044
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% define cdata properties %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046
0047 function c = init()
0048     c.name      = 'None';
0049     c.y         = [];
0050     c.x         = {};
0051     c.xunits    = 'N/A';
0052     c.yunits    = 'N/A';
0053     c.version   = VERSION;
0054     c.created   = time;
0055     c = class(c, 'cdata');
0056 end
0057
0058 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Create cdata object %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0059
0060 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% val = cdata() %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0061 % create default cdata object
0062 if nargin == 0
0063     c = init();
0064
0065 elseif nargin == 1
0066
0067     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Create from XML fragment %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0068     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0069         c = fromxml(varargin{1});
0070     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% From File %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0071     elseif ischar(varargin{1})
0072
0073         filename = varargin{1};
0074         [path, name, ext, vers] = fileparts(filename);
0075         switch ext
0076             case '.mat'
0077                 c = load(filename);
0078                 c = c.a;
0079             case '.xml'
0080                 c = xmlparse(cdata, filename);
0081             otherwise
0082                 error('### Unknown file type.');
```

```

0087
0088     %%%%%%%%%%% val = cdata(struct) %%%%%%%%%%%
0089 elseif isstruct(varargin{1})
0090
0091     c = init();
0092
0093     fields = fieldnames(varargin{1});
0094     for ii = 1:length(fields)
0095         field = fields{ii};
0096
0097         %%% created -> time-object
0098         if strcmp(field, 'created')
0099             created = varargin{1}.created;
0100             if isstruct(created)
0101                 created = time(created);
0102             end
0103             c.created = created;
0104         %%% All other
0105         else
0106             try
0107                 c.(field) = varargin{1}.(field);
0108             catch
0109                 error('### The field '%s' in the struct is not a cdata property.',
field)
0110             end
0111         end
0112     end
0113
0114     %%%%%%%%%%% val = cdata(plist) %%%%%%%%%%%
0115 elseif isa(varargin{1}, 'plist')
0116
0117     c = init();
0118
0119     % get possible parameters
0120     fcn = find(varargin{1}, 'fcn');
0121
0122     y = find(varargin{1}, 'vals');
0123     if isempty(y)
0124         y = find(varargin{1}, 'y');
0125     end
0126
0127     % check which to use
0128     if ~isempty(fcn)
0129         % do function constructor
0130         y = eval(fcn);
0131     elseif ~isempty(y)
0132         % do vals constructor
0133         %- nothing to do here.
0134     else
0135         error('### unknown constructor method for cdata.');
```

```

0138     c.name = fcn;
0139     c.y     = y;
0140
0141     %%%%%%%%%%% val = cdata(vals) %%%%%%%%%%%
0142     %%%%%%%%%%% val = cdata(y) %%%%%%%%%%%
0143 else
0144
0145     c      = init();
0146     c.name = 'Data';
0147     c.y     = varargin{1};
0148
0149 end
0150
0151
0152 %%%%%%%%%%% val = cdata(N,val) %%%%%%%%%%%
0153 elseif nargin == 2
0154
0155 %%%%%%%%%%% From DATABASE
0156 if isa(varargin{1}, 'database')
0157     c = retrieve(varargin{1}, varargin{2:end});
0158 else
0159     c = init();
0160     c.name = 'Constant';
0161     c.y     = ones(varargin{1},1)*varargin{2};
0162 end
0163 else
0164     error('### Unknown number of constructor arguments. ');
0165 end
0166
0167 % check data
0168 % c = checkCdata(c);
0169 end
0170
0171
0172 %-----
0173 % check vector has right dimensions
0174 function c = checkCdata(c)
0175
0176 x = c.y;
0177 if size(x, 1) < size(x, 2)
0178     x = x.';
0179 end
0180
0181 c = set(c, 'y', x);
0182 end
0183

```


[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_HFT169D.m

win_HFT169D

PURPOSE

WIN_HFT169D return HFT169D window sample for input z.

SYNOPSIS

```
function w = win_HFT169D(z)
```

DESCRIPTION

```
WIN_HFT169D return HFT169D window sample for input z.
```

```
M Hewitson 18-05-07
```

```
$Id: win_HFT169D.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
```

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_HFT169D(z)
0002
0003 % WIN_HFT169D return HFT169D window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT169D.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 z = z * 2 * pi;
0011 v = (1 - 1.97441843 * cos (z) ...
0012      + 1.65409889 * cos (2 * z) - 0.95788187 * cos (3 * z) + ...
0013      0.33673420 * cos (4 * z) - 0.06364622 * cos (5 * z) + ...
0014      0.00521942 * cos (6 * z) - 0.00010599 * cos (7 * z));
0015
0016 % Make window struct
0017 w.name      = 'HFT169D';
0018 w.alpha     = 0;
0019 w.psll      = 169.5;
0020 w.rov       = 81.2;
0021 w.nenbw     = 4.8347;
0022 w.w3db      = 4.7588;
0023 w.flatness  = 0.0017;
0024 w.ws        = sum(v);
0025 w.ws2       = sum(v.*v);
0026 w.win       = v;
0027
0028 % END
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > kaiser_alpha.m

kaiser_alpha

PURPOSE

KAISER_ALPHA returns the alpha parameter that gives the required input

SYNOPSIS

```
function alpha = kaiser_alpha(req_psl)
```

DESCRIPTION

KAISER_ALPHA returns the alpha parameter that gives the required input PSL.

Taken from C code of Gerhard Heinzel:

```
Compute the parameter alpha of Kaiser windows
from the required PSL [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 PSL > 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

```
0001 function alpha = kaiser_alpha(req_psl)
0002
```

```
0003 % KAISER_ALPHA returns the alpha parameter that gives the required input
0004 % PSLI.
0005 %
0006 % Taken from C code of Gerhard Heinzel:
0007 %
0008 %     Compute the parameter alpha of Kaiser windows
0009 %     from the required PSLI [dB]. Best-fit polynomial
0010 %     was obtained from 180 data points between alpha=1
0011 %     and alpha=9.95. Maximum error is 0.05
0012 %     Maximum error for PSLI > 30 dB is 0.02
0013 %
0014 % M Hewitson 19-05-07
0015 %
0016 % $Id: kaiser_alpha.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0017 %
0018
0019
0020 a0 = -0.0821377;
0021 a1 = 4.71469;
0022 a2 = -0.493285;
0023 a3 = 0.0889732;
0024
0025 x = req_psl / 100;
0026 alpha = (((((a3 * x) + a2) * x) + a1) * x + a0);
0027
0028
0029 % END
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_HFT196D.m

win_HFT196D

PURPOSE

WIN_HFT196D return HFT196D window sample for input z.

SYNOPSIS

```
function w = win_HFT196D(z)
```

DESCRIPTION

WIN_HFT196D return HFT196D window sample for input z.

M Hewitson 18-05-07

\$Id: win_HFT196D.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_HFT196D(z)
0002
0003 % WIN_HFT196D return HFT196D window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT196D.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = (1 - 1.979280420 * cos (z) ...
```

```
0012      + 1.710288951 * cos (2 * z) - 1.081629853 * cos (3 * z) + ...
0013      0.448734314 * cos (4 * z) - 0.112376628 * cos (5 * z) + ...
0014      0.015122992 * cos (6 * z) - 0.000871252 * cos (7 * z) + ...
0015      0.000011896 * cos (8 * z));
0016
0017 % Make window struct
0018 w.name      = 'HFT196D';
0019 w.alpha     = 0;
0020 w.psl1      = 196.2;
0021 w.rov       = 82.3;
0022 w.nenbw    = 5.1134;
0023 w.w3db     = 5.0308;
0024 w.flatness = 0.0013;
0025 w.ws       = sum(v);
0026 w.ws2      = sum(v.*v);
0027 w.win      = v;
0028
0029 % END
```

[Home](#) > [classes](#) > [@cdata](#) > char.m

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 1.3 2007/11/14 18:18:36 ingo 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.
- [eq](#) EQ overloads the == operator for cdata objects.
- [get](#) GET get csdata properties.
- [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](#) 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 vectors in tsdata.
- [subsref](#) SUBSREF Define field name indexing for cdata objects.
- [xmlparse](#) XMLPARSE creates a cdata from a cdata xml file.

SOURCE CODE

```

0001 function pstr = char(p, varargin)
0002 % CHAR convert a cdata object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a cdata object into a string.
0007 %
0008 % VERSION: $Id: char.m,v 1.3 2007/11/14 18:18:36 ingo Exp $
0009 %
0010 % HISTORY: 04-02-2007 M Hewitson
0011 %           Creation
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014
0015 VERSION= '$Id: char.m,v 1.3 2007/11/14 18:18:36 ingo Exp $';
0016
0017 % Check if this is a call for parameters
0018 if nargin == 2
0019     if isa(p, 'cdata') && ischar(varargin{1})
0020         in = char(varargin{1});
0021         if strcmp(in, 'Params')
0022             pstr = plist();
0023             return
0024         elseif strcmp(in, 'Version')
0025             pstr = VERSION;
0026             return
0027         end
0028     end
0029 end
0030
0031 pstr = get(p, 'name');
0032
0033
0034 % END

```


[Home](#) > [classes](#) > [@specwin](#) > [private](#) > kaiser_flatness.m

kaiser_flatness

PURPOSE

KAISER_FLATNESS returns the flatness in dB of the central bin of a kaiser

SYNOPSIS

function flatness = kaiser_flatness(alpha)

DESCRIPTION

KAISER_FLATNESS returns the flatness in dB of the central bin of a kaiser window with parameter alpha.

Taken from C code of Gerhard Heinzel:

Compute the flatness in the central bin [dB]
of Kaiser windows from the parameter alpha.
Best-fit polynomial was obtained from 180 data
points between alpha=1 and alpha=9.95.
Maximum error is 0.013 dB.

M Hewitson 19-05-07

\$Id: kaiser_flatness.m,v 1.1 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

```
0001 function flatness = kaiser_flatness(alpha)
0002
0003 % KAISER_FLATNESS returns the flatness in dB of the central bin of a kaiser
0004 % window with parameter alpha.
```

```
0005 %
0006 % Taken from C code of Gerhard Heinzel:
0007 %
0008 %     Compute the flatness in the central bin [dB]
0009 %     of Kaiser windows from the parameter alpha.
0010 %     Best-fit polynomial was obtained from 180 data
0011 %     points between alpha=1 and alpha=9.95.
0012 %     Maximum error is 0.013 dB.
0013 %
0014 % M Hewitson 19-05-07
0015 %
0016 % $Id: kaiser_flatness.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0017 %
0018
0019 a0      = 0.141273;
0020 a1      = 0.262425;
0021 a2      = 0.00642551;
0022 a3      = -0.000405621;
0023 x       = alpha;
0024 flatness = -1. / (((((a3 * x) + a2) * x) + a1) * x + a0);
0025
0026 % END
```

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 1.3 2007/12/06 17:09:15 ingo 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

```

0001 function filt = redesign(filt, fs)
0002 % REDESIGN redesign the input filter to work for the given sample rate.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: REDESIGN redesign the input filter to work for the given sample
0007 %               rate.
0008 %
0009 % CALL:         filt = redesign(filt, fs)
0010 %
0011 % INPUT:        filt - input (mfir) filter
0012 %               fs   - new sample rate
0013 %
0014 % OUTPUT:       filt - new output filter
0015 %
0016 % VERSION:      $Id: redesign.m,v 1.3 2007/12/06 17:09:15 ingo Exp $
0017 %
0018 % HISTORY:      04-04-2007 M Hewitson
0019 %               Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 VERSION = '$Id: redesign.m,v 1.3 2007/12/06 17:09:15 ingo Exp $';
0024
0025 % Check if this is a call for parameters
0026 if nargin == 2
0027     if isa(filt, 'mfir') && ischar(fs)
0028         in = char(fs);
0029         if strcmp(in, 'Params')
0030             filt = plist;
0031             return
0032         elseif strcmp(in, 'Version')
0033             filt = VERSION;
0034             return
0035         end
0036     end
0037 end
0038
0039 disp(sprintf('*** re-designing filter for fs=%2.2f Hz...', fs));
0040 fpl = get(filt, 'plist');
0041 type = find(fpl, 'type');
0042 pzm = find(fpl, 'pzmodel');
0043
0044 if strcmp(type, 'highpass') || ...
0045    strcmp(type, 'lowpass') || ...
0046    strcmp(type, 'bandpass') || ...
0047    strcmp(type, 'bandreject')
0048
0049     disp('    - re-designing standard filter.');
```

```
0050     try
0051         fpl = pset(fpl, 'fs', fs);
0052         disp('    - setting new fs.');
```

0053 catch

```
0054         fpl = append(fpl, 'fs', fs);
0055         disp('    - appending parameter fs.');
```

0056 end

```
0057     filtnew = mfir(fpl);
0058     filtnew = set(filtnew, 'histout', get(filt, 'histout'));
0059     filt = filtnew;
0060 elseif ~isempty(pzm)
0061
0062     disp('    - re-designing pzmodel filter.');
```

0063 filtnew = mfir(plist([param('pzmodel', pzm) param('fs', fs)]));

```
0064     filtnew = set(filtnew, 'histout', get(filt, 'histout'));
0065     filt = filtnew;
0066
0067 else
0068
0069     warning('!!! un-recognised input filter type. Can''t redesign.');
```

0070

0071

0072

0073 end

0074

0075

0076

0077

0078 % END

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_HFT223D.m

win_HFT223D

PURPOSE

WIN_HFT223D return HFT223D window sample for input z.

SYNOPSIS

```
function w = win_HFT223D(z)
```

DESCRIPTION

WIN_HFT223D return HFT223D window sample for input z.

M Hewitson 18-05-07

\$Id: win_HFT223D.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_HFT223D(z)
0002
0003 % WIN_HFT223D return HFT223D window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT223D.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = (1 - 1.98298997309 * cos(z) + ...
0012      1.75556083063 * cos (2 * z) - 1.19037717712 * cos (3 * z) + ...
```

```
0013      0.56155440797 * cos (4 * z) - 0.17296769663 * cos (5 * z) + ...
0014      0.03233247087 * cos (6 * z) - 0.00324954578 * cos (7 * z) + ...
0015      0.00013801040 * cos (8 * z) - 0.00000132725 * cos (9 * z));
0016
0017 % Make window struct
0018 w.name      = 'HFT223D';
0019 w.alpha     = 0;
0020 w.psll      = 223;
0021 w.rov       = 83.3;
0022 w.nenbw     = 5.3888;
0023 w.w3db      = 5.3;
0024 w.flatness  = -0.0011;
0025 w.ws        = sum(v);
0026 w.ws2       = sum(v.*v);
0027 w.win       = v;
0028
0029 % END
```

[Home](#) > [classes](#) > [@fsdata](#) > char.m

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 1.3 2007/11/14 18:18:36 ingo 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.
- [eq](#) EQ overloads the == operator for fsdata objects.
- [get](#) GET get a fsdata property.

- **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** SET set a fsdata property.
- **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.
- **xmlparse** XMLPARSE creates a fsdata from a fsdata xml file.

SOURCE CODE

```

0001 function pstr = char(p, varargin)
0002 % CHAR convert a param object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a param object into a string.
0007 %
0008 % CALL:          string = char(fsd)
0009 %
0010 % INPUTS:        fsd      - frequency-series object
0011 %
0012 % OUTPUTS:       string - a string which represent a frequency-series object
0013 %
0014 % VERSION:       $Id: char.m,v 1.3 2007/11/14 18:18:36 ingo Exp $
0015 %
0016 % HISTORY:       04-02-2007 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION= '$Id: char.m,v 1.3 2007/11/14 18:18:36 ingo Exp $';
0022
0023 % Check if this is a call for parameters
0024 if nargin == 2
0025     if isa(p, 'fsdata') && ischar(varargin{1})
0026         in = char(varargin{1});
0027         if strcmp(in, 'Params')
0028             pstr = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             pstr = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 pstr = get(p, 'name');
0038

```

0039

0040 % END

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > kaiser_nenbw.m

kaiser_nenbw

PURPOSE

KAISER_NENBW returns the normalized noise-equivalent bandwidth for a

SYNOPSIS

```
function nenbw = kaiser_nenbw(alpha)
```

DESCRIPTION

KAISER_NENBW returns the normalized noise-equivalent bandwidth for a kaiser window with parameter alpha.

Take from C code of Gerhard Heinzel:

```
Compute the 'normalized noise-equivalent bandwidth'
(NENBW) [bins] of Kaiser windows from the parameter alpha.
Best-fit polynomial was obtained from 180 data
points between alpha=1 and alpha=9.95.
Maximum error is 0.007 bins
NOTE that NENBW can be computed precisely
from the actual time-domain window values
```

M Hewitson 19-05-07

\$Id: kaiser_nenbw.m,v 1.1 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

```
0001 function nenbw = kaiser_nenbw(alpha)
0002
0003 % KAISER_NENBW returns the normalized noise-equivalent bandwidth for a
0004 % kaiser window with parameter alpha.
0005 %
0006 % Take from C code of Gerhard Heinzel:
0007 %
0008 %     Compute the 'normalized noise-equivalent bandwidth'
0009 %     (NENBW) [bins] of Kaiser windows from the parameter alpha.
0010 %     Best-fit polynomial was obtained from 180 data
0011 %     points between alpha=1 and alpha=9.95.
0012 %     Maximum error is 0.007 bins
0013 %     NOTE that NENBW can be computed precisely
0014 %     from the actual time-domain window values
0015 %
0016 % M Hewitson 19-05-07
0017 %
0018 % $Id: kaiser_nenbw.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0019 %
0020
0021 a0      = 0.768049;
0022 a1      = 0.411986;
0023 a2      = -0.0264817;
0024 a3      = 0.000962211;
0025 x       = alpha;
0026 nenbw = (((((a3 * x) + a2) * x) + a1) * x + a0);
0027
0028 % END
```

[Home](#) > [classes](#) > [@miir](#) > redesign.m

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 1.8 2007/12/06 17:09:15 ingo 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

```

0001 function filt = redesign(filt, fs)
0002 % REDESIGN redesign the input filter to work for the given sample rate.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: REDESIGN redesign the input filter to work for the given sample
0007 %               rate.
0008 %
0009 % CALL:         filt = redesign(filt, fs)
0010 %
0011 % INPUT:        filt - input (miir) filter
0012 %               fs   - new sample rate
0013 %
0014 % OUTPUT:       filt - new output filter
0015 %
0016 % VERSION:      $Id: redesign.m,v 1.8 2007/12/06 17:09:15 ingo Exp $
0017 %
0018 % HISTORY:      04-04-2007 M Hewitson
0019 %               Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 VERSION = '$Id: redesign.m,v 1.8 2007/12/06 17:09:15 ingo Exp $';
0024
0025 % Check if this is a call for parameters
0026 if nargin == 2
0027     if isa(filt, 'miir') && ischar(fs)
0028         in = char(fs);
0029         if strcmp(in, 'Params')
0030             filt = plist;
0031             return
0032         elseif strcmp(in, 'Version')
0033             filt = VERSION;
0034             return
0035         end
0036     end
0037 end
0038
0039 disp(sprintf('*** re-designing filter for fs=%2.2f Hz...', fs));
0040 fpl = get(filt, 'plist');
0041 type = find(fpl, 'type');
0042 pzm = find(fpl, 'pzmodel');
0043
0044 if strcmp(type, 'highpass') || ...
0045    strcmp(type, 'lowpass') || ...
0046    strcmp(type, 'bandpass') || ...
0047    strcmp(type, 'bandreject')
0048
0049     disp('    - re-designing standard filter.');
```

```
0050     try
0051         fpl = pset(fpl, 'fs', fs);
0052         disp('    - setting new fs.');
```

0053 catch

```
0054         fpl = append(fpl, 'fs', fs);
0055         disp('    - appending parameter fs.');
```

0056 end

```
0057     filtnew = miir(fpl);
0058     filtnew = set(filtnew, 'histin', get(filt, 'histin'));
0059     filtnew = set(filtnew, 'histout', get(filt, 'histout'));
0060     filt = filtnew;
0061 elseif ~isempty(pzm)
0062
0063     disp('    - re-designing pzmodel filter.');
```

0064 filtnew = miir(plist([param('pzmodel', pzm) param('fs', fs)]));

```
0065     filtnew = set(filtnew, 'histin', get(filt, 'histin'));
0066     filtnew = set(filtnew, 'histout', get(filt, 'histout'));
0067     filt = filtnew;
0068
0069 else
0070
0071     warning('!!! un-recognised input filter type. Can''t redesign.');
```

0072

0073

0074

0075 end

```
0076
0077
0078
0079
0080 % END
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_HFT248D.m

win_HFT248D

PURPOSE

WIN_HFT248D return HFT248D window sample for input z.

SYNOPSIS

```
function w = win_HFT248D(z)
```

DESCRIPTION

WIN_HFT248D return HFT248D window sample for input z.

M Hewitson 18-05-07

\$Id: win_HFT248D.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_HFT248D(z)
0002
0003 % WIN_HFT248D return HFT248D window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT248D.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = (1 - 1.985844164102 * cos(z) + ...
0012      1.791176438506 * cos (2 * z) - 1.282075284005 * cos (3 * z) + ...
0013      0.667777530266 * cos (4 * z) - 0.240160796576 * cos (5 * z) + ...
```



```
0014      0.056656381764 * cos (6 * z) - 0.008134974479 * cos (7 * z) + ...
0015      0.000624544650 * cos (8 * z) - 0.000019808998 * cos (9 * z) + ...
0016      0.000000132974 * cos (10 * z));
0017
0018 % Make window struct
0019 w.name      = 'HFT248D';
0020 w.alpha     = 0;
0021 w.psll      = 248;
0022 w.rov       = 84.1;
0023 w.nenbw     = 5.6512;
0024 w.w3db      = 5.5567;
0025 w.flatness  = 0.0009;
0026 w.ws        = sum(v);
0027 w.ws2       = sum(v.*v);
0028 w.win       = v;
0029
0030 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@history](#) > char.m

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 1.3 2007/12/04 18:16:11 ingo 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.
- [eq](#) EQ overloads the == operator for history objects.
- [get](#) GET get history object properties.
- [getNode](#) GETNODES converts a history object to a nodes structure suitable for plotting as a tree.
- [getNode_plot](#) GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.
- [hist2dot](#) HIST2DOT converts a history object to a 'DOT' file suitable for processing with graphviz

- **hist2m** HIST2M writes a new m-file that reproduces the analysis described in the history object.
- **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.
- **xmlparse** XMLPARSE creates a history from a history xml file.

SOURCE CODE

```

0001 function pstr = char(p, varargin)
0002 % CHAR convert a param object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a param object into a string.
0007 %
0008 % CALL:          string = char(hist)
0009 %
0010 % VERSION:       $Id: char.m,v 1.3 2007/12/04 18:16:11 ingo Exp $
0011 %
0012 % HISTORY:       04-02-2007 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION= '$Id: char.m,v 1.3 2007/12/04 18:16:11 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(p, 'history') && ischar(varargin{1})
0022         in = char(varargin{1});
0023         if strcmp(in, 'Params')
0024             pstr = plist();
0025             return
0026         elseif strcmp(in, 'Version')
0027             pstr = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 pstr = get(p, 'name');
0034
0035 % END

```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > kaiser_rov.m

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

```
0001 function rov = kaiser_rov(alpha)
0002
0003 % KAISER_ROV returns the recommended overlap for a Kaiser window with
0004 % parameter alpha.
0005 %
0006 % Taken from C code of Gerhard Heinzel:
0007 %
0008 %     Compute the 'recommended overlap' (ROV) [%] of Kaiser windows
0009 %     from the parameter alpha. Best-fit polynomial
0010 %     was obtained from 180 data points between alpha=1
0011 %     and alpha=9.95. Maximum error is 1.5%, mainly due
0012 %     to insufficient precision in the data points
0013 %
0014 % M Hewitson 19-05-07
0015 %
0016 % $Id: kaiser_rov.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0017 %
0018
0019
0020 a0 = 0.0061076;
0021 a1 = 0.00912223;
0022 a2 = -0.000925946;
0023 a3 = 4.42204e-05;
0024 x = alpha;
0025 rov = 100 - 1 / (((((a3 * x) + a2) * x) + a1) * x + a0);
0026
0027 % END
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_HFT70.m

win_HFT70

PURPOSE

WIN_HFT70 return HFT70 window sample for input z.

SYNOPSIS

function w = win_HFT70(z)

DESCRIPTION

WIN_HFT70 return HFT70 window sample for input z.

M Hewitson 18-05-07

\$Id: win_HFT70.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_HFT70(z)
0002
0003 % WIN_HFT70 return HFT70 window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT70.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 z = z * 2 * pi;
0011 v = (1 - 1.90796 * cos (z) + ...
0012      1.07349 * cos (2 * z) - 0.18199 * cos (3 * z));
0013
0014 % Make window struct
0015 w.name      = 'HFT70';
0016 w.alpha     = 0;
0017 w.psll      = 70.4;
0018 w.rov       = 72.2;
0019 w.nenbw     = 3.4129;
0020 w.w3db      = 3.3720;
0021 w.flatness  = -0.0065;
0022 w.ws        = sum(v);
0023 w.ws2       = sum(v.*v);
0024 w.win       = v;
0025
0026 % END
```

[Home](#) > [classes](#) > [@mfir](#) > char.m

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 1.3 2007/12/06 17:09:15 ingo 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.
- [eq](#) EQ overloads the == operator for mfir objects.
- [get](#) GET get mfir properties.
- [ne](#) NE overloads the ~= operator for mfir objects.
- [redesign](#) REDESIGN redesign the input filter to work for the given sample rate.
- [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.
- [xmlparse](#) XMLPARSE creates a mfir filter from a mfir xml file.

SOURCE CODE

```

0001 function pstr = char(w, varargin)
0002 % CHAR convert a mfir object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a mfir object into a string.
0007 %
0008 % CALL:          string = char(mfir)
0009 %
0010 % VERSION:       $Id: char.m,v 1.3 2007/12/06 17:09:15 ingo Exp $
0011 %
0012 % HISTORY:       04-02-2007 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION= '$Id: char.m,v 1.3 2007/12/06 17:09:15 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(w, 'mfir') && ischar(varargin{1})
0022         in = char(varargin{1});
0023         if strcmp(in, 'Params')
0024             pstr = plist();
0025             return
0026         elseif strcmp(in, 'Version')
0027             pstr = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 pstr = [];
0034 pstr = [pstr w.name];
0035 pstr = [pstr ', fs=' num2str(w.fs)];
0036 pstr = [pstr ', ntaps=' num2str(w.ntaps)];
0037 pstr = [pstr ', a=' num2str(w.a)];
0038 pstr = [pstr ', gain=' num2str(w.gain)];
0039
0040 pstr = strrep(pstr, '_', '\_');
0041
0042 % END

```


[Home](#) > [classes](#) > [@specwin](#) > [private](#) > kaiser_w3db.m

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 [^](#)

```
0001 function w3db = kaiser_w3db(alpha)
```

```
0002
0003 % KAISER_W3DB returns the 3dB bandwidth in bins of a kaiser window with
0004 % parameter alpha.
0005 %
0006 % Taken from C code of Gerhard Heinzel:
0007 %
0008 %     Compute the 3dB bandwidth (W3db) [bins]
0009 %     of Kaiser windows from the parameter alpha.
0010 %     Best-fit polynomial was obtained from 180 data
0011 %     points between alpha=1 and alpha=9.95.
0012 %     Maximum error is 0.006 bins.
0013 %
0014 % M Hewitson 19-05-07
0015 %
0016 % $Id: kaiser_w3db.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0017 %
0018
0019 a0    = 0.757185;
0020 a1    = 0.377847;
0021 a2    = -0.0238342;
0022 a3    = 0.00086012;
0023 x     = alpha;
0024 w3db  = (((((a3 * x) + a2) * x) + a1) * x + a0);
0025
0026 % END
```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > repogui.m

repogui

PURPOSE

REPOGUI a GUI to interact with an LTPDA Repository.

SYNOPSIS

```
function varargout = repogui(varargin)
```

DESCRIPTION

REPOGUI a GUI to interact with an LTPDA Repository.

M Hewitson

CROSS-REFERENCE INFORMATION

This function calls:

- [uitabpanel](#) UITABPANEL adds a group of tabbed uipanel container objects to a figure.

This function is called by:

SUBFUNCTIONS

- [function panel = create_quicklook_panel\(panel, Gproperties\)](#)
- [function panel = create_retrieve_panel\(panel, Gproperties\)](#)
- [function panel = create_query_panel\(panel, Gproperties\)](#)
- [function panel = create_submit_panel\(panel, Gproperties\)](#)
- [function panel = create_connection_panel\(panel, Gproperties\)](#)

SOURCE CODE

```
0001 function varargout = repogui(varargin)
0002
0003 % REPOGUI a GUI to interact with an LTPDA Repository.
0004 %
0005 % M Hewitson
0006 %
0007
0008 % addpath('callbacks');
```

```

0009
0010 %% Check if I exist already
0011 id = findobj('Tag', 'LTPDARepomainfig');
0012 if ~isempty(id)
0013     figure(id)
0014     return
0015 end
0016
0017 %% Some initial setup
0018
0019
0020
0021 Gproperties.Gcol      = [240 240 240]/255;
0022 Gproperties.Gwidth    = 800;
0023 Gproperties.Gheight   = 600;
0024 Gproperties.Gborder   = 10;
0025 fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
0026
0027 Gproperties.Screen    = get(0, 'screensize');
0028 Gproperties.Gposition = [Gproperties.Screen(3)/2-Gproperties.Gwidth/2 ...
0029     Gproperties.Screen(4)/2-Gproperties.Gheight/2 ...
0030     Gproperties.Gwidth...
0031     Gproperties.Gheight];
0032
0033 % Initialize and hide the GUI as it is being constructed.
0034 mainfig = figure('Name', 'LTPDA Repository GUI',...
0035     'NumberTitle', 'off',...
0036     'Visible', 'off',...
0037     'Position', Gproperties.Gposition,...
0038     'Color', Gproperties.Gcol,...
0039     'ToolBar', 'none',...
0040     'MenuBar', 'none',...
0041     'Resize', 'off',...
0042     'Tag', 'LTPDARepomainfig');
0043
0044 % Set mainfig callbacks
0045 set(mainfig, 'CloseRequestFcn', {'cb_mainfigClose', mainfig});
0046
0047 % Set Application data
0048 setappdata(mainfig, 'Gproperties', Gproperties);
0049 setappdata(mainfig, 'connection', []);
0050 setappdata(mainfig, 'Nconditions', 0);
0051 setappdata(mainfig, 'tableSelected', 1);
0052
0053 %% Add components
0054
0055 Tleft      = Gproperties.Gborder/Gproperties.Gwidth;
0056 Theight    = 1-2*Gproperties.Gborder/Gproperties.Gheight;
0057 Tbottom    = (1 - Gproperties.Gborder/Gproperties.Gheight - Theight);
0058 Twidth     = 1-2*Gproperties.Gborder/Gproperties.Gwidth;
0059

```

```

0060 htab = uitabpanel(...
0061     'Parent',mainfig,...
0062     'Style','lefttop',...
0063     'Units','normalized',...
0064     'Position',[Tleft Tbottom Twidth Theight],...
0065     'FontSize', fontsize,...
0066     'FrameBackgroundColor',Gproperties.Gcol,...
0067     'FrameBorderType','etchedout',...
0068     'Title',{'Connection','Submit','Query','Retrieve'},...%, 'Quick Look'},...
0069     'HorizontalAlignment','left',...
0070     'FontWeight','bold',...
0071     'TitleBackgroundColor',Gproperties.Gcol,...
0072     'TitleForegroundColor','k',...
0073     'TitleHighlightColor',[1 0 0],...
0074     'PanelBackgroundColor',Gproperties.Gcol,...
0075     'PanelBorderType','line');
0076
0077 hpanel = getappdata(htab,'panels');
0078 setappdata(mainfig,'panels', hpanel);
0079
0080 %----- Connection Panel -----
0081 cp = create_connection_panel(hpanel(1), Gproperties);
0082 %----- Submit Panel -----
0083 sp = create_submit_panel(hpanel(2), Gproperties);
0084 %----- Query Panel -----
0085 qp = create_query_panel(hpanel(3), Gproperties);
0086 %----- Query Panel -----
0087 rp = create_retrieve_panel(hpanel(4), Gproperties);
0088 %----- Quick Look Panel -----
0089 % rp = create_quicklook_panel(hpanel(5), Gproperties);
0090
0091
0092 %% Start the GUI
0093
0094 % Make the GUI visible.
0095 set(mainfig,'Visible','on')
0096
0097 h = findobj(mainfig, 'Tag', 'statusTxt');
0098 get(h, 'String')
0099
0100 end
0101
0102 %-----
0103 % Create the quick look panel
0104 %
0105 function panel = create_quicklook_panel(panel, Gproperties)
0106
0107 Ppos      = get(panel, 'Position');
0108 mainfig = findobj('Tag', 'LTPDARepomainfig');
0109

```

```

0110 %-- Text string
0111 Sleft    = 10;
0112 Sheight  = 20;
0113 Sbottom  = Ppos(2) + Ppos(4) - 20 - Gproperties.Gborder - Sheight;
0114 Swidth   = 200;
0115 fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
0116 sth = uicontrol(panel, 'Style', 'text', ...
0117     'String', 'LTPDA Objects', ...
0118     'FontSize', fontsize, ...
0119     'BackgroundColor', Gproperties.Gcol, ...
0120     'HorizontalAlignment', 'center', ...
0121     'Position', [Sleft Sbottom Swidth Sheight]);
0122
0123 %-- Workspace list of LTPDA objects
0124 Lleft    = 10;
0125 Lheight  = 350;
0126 Lbottom  = Ppos(2) + Ppos(4) - 40 - Gproperties.Gborder - Lheight;
0127 Lwidth   = Swidth;
0128
0129 lbh = uicontrol(panel, 'Style', 'listbox', ...
0130     'String', {' '}, ...
0131     'Value', 1, ...
0132     'BackgroundColor', 'w', ...
0133     'FontSize', fontsize, ...
0134     'Max', 1000, ...
0135     'Position', [Lleft Lbottom Lwidth Lheight], ...
0136     'Tag', 'workspaceObjsListQL');
0137
0138 % Set callback
0139 set(lbh, 'Callback', {'cb_workspaceObjsListQL', lbh, mainfig});
0140
0141 % fill list
0142 objs = getWorkspaceObjs();
0143 setWorkspaceObjsListQL(objs);
0144
0145 %-- Refresh button
0146 pos    = get(lbh, 'Position');
0147 Bleft  = Lleft;
0148 Bheight = 20;
0149 Bwidth  = 100;
0150 Bbottom = pos(2) - Bheight - 10;
0151 pbh = uicontrol(panel, 'Style', 'pushbutton', 'String', 'Refresh list', ...
0152     'Position', [Bleft Bbottom Bwidth Bheight ], ...
0153     'Tag', 'refreshListBtnQL');
0154 set(pbh, 'Callback', {'cb_refreshObjsListQL', pbh, mainfig});
0155
0156
0157 %-- Info panel
0158 pos = get(lbh, 'Position');
0159 Sleft  = 10 + pos(1) + pos(3);
0160 Sheight = 150;

```



```

0161 Sbottom = Ppos(2) + Ppos(4) - 40 - Gproperties.Gborder - Sheight;
0162 Swidth  = 550;
0163 iph = uicontrol(panel,'Style','edit',...
0164     'String','',...
0165     'FontSize', fontsize,...
0166     'BackgroundColor', 'w',...
0167     'ForegroundColor', 'b',...
0168     'HorizontalAlignment', 'left',...
0169     'Enable', 'on',...
0170     'Max', 1000,...
0171     'Position',[Sleft Sbottom Swidth Sheight],...
0172     'Tag', 'ltpdaObjInfoQL');
0173
0174 %-- Text string
0175 pos = get(iph, 'Position');
0176 Sheight = 20;
0177 Sbottom = pos(2) - Sheight - 20;
0178 Swidth  = 300;
0179 sth = uicontrol(panel,'Style','text',...
0180     'String','Class Methods',...
0181     'FontSize', fontsize,...
0182     'BackgroundColor', Gproperties.Gcol,...
0183     'HorizontalAlignment', 'left',...
0184     'Position',[Sleft Sbottom Swidth Sheight]);
0185
0186 %-- List of class methods
0187 pos = get(lbh, 'Position');
0188 Lleft  = pos(1) + pos(3) + 10;
0189 Lheight = 150;
0190 Lbottom = pos(2);
0191 Lwidth  = 150;
0192
0193 lbh = uicontrol(panel,'Style','listbox',...
0194     'String',{' '},...
0195     'Value',1,...
0196     'BackgroundColor', 'w',...
0197     'FontSize', fontsize,...
0198     'Max', 1000,...
0199     'Position',[Lleft Lbottom Lwidth Lheight],...
0200     'Tag', 'classMethodList');
0201
0202 % Set callback
0203 set(lbh, 'Callback', {'cb_classMethodList', lbh, mainfig});
0204
0205 % fill list
0206 fillClassMethodsList();
0207
0208 %----- Help panel
0209 pos = get(lbh, 'Position');
0210 Sleft  = pos(1) + pos(3) + 10;
0211 Sheight = 150;

```

```

0212 Sbottom = pos(2);
0213 Swidth  = 550;
0214 sth = uicontrol(panel,'Style','edit',...
0215     'String','',...
0216     'FontSize', fontsize,...
0217     'BackgroundColor', 'w',...
0218     'ForegroundColor', 'b',...
0219     'HorizontalAlignment', 'left',...
0220     'Enable', 'on',...
0221     'Max', 1000,...
0222     'Position',[Sleft Sbottom Swidth Sheight],...
0223     'Tag', 'methodInfoPanel');
0224
0225
0226 end
0227
0228 %-----
0229 % Create the retrieve panel
0230 %
0231 function panel = create_retrieve_panel(panel, Gproperties)
0232
0233 Ppos      = get(panel, 'Position');
0234 mainfig = findobj('Tag', 'LTPDAREpomainfig');
0235
0236 %-- Text string
0237 Sleft     = 10;
0238 Sheight  = 20;
0239 Sbottom = Ppos(2) + Ppos(4) - 20 - Gproperties.Gborder - Sheight;
0240 Swidth   = 300;
0241 fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
0242 Osth = uicontrol(panel,'Style','text',...
0243     'String','Retrieve object IDs',...
0244     'FontSize', fontsize,...
0245     'BackgroundColor', Gproperties.Gcol,...
0246     'HorizontalAlignment', 'left',...
0247     'Position',[Sleft Sbottom Swidth Sheight]);
0248
0249 %-- ID entry
0250 pos      = get(Osth, 'Position');
0251 Sleft    = 10;
0252 Sheight  = 250;
0253 Swidth   = 250;
0254 Sbottom = pos(2) - 20 - Sheight;
0255
0256 sth = uicontrol(panel,'Style','edit',...
0257     'String','',...
0258     'FontSize', fontsize,...
0259     'BackgroundColor', 'w',...
0260     'Max', 10,...
0261     'HorizontalAlignment', 'left',...
0262     'Position',[Sleft Sbottom Swidth Sheight],...

```

```

0263     'Tag', 'retrieveIDsTxt');
0264
0265 %-- Prefix entry
0266 pos      = get(sth, 'Position');
0267 Sleft     = 10;
0268 Sheight   = 20;
0269 Sbottom   = pos(2) - 20 - Sheight;
0270 Swidth    = 50;
0271 sth = uicontrol(panel, 'Style', 'text', ...
0272     'String', 'Prefix', ...
0273     'FontSize', fontsize, ...
0274     'BackgroundColor', Gproperties.Gcol, ...
0275     'HorizontalAlignment', 'left', ...
0276     'Position', [Sleft Sbottom Swidth Sheight]);
0277
0278
0279 Sleft     = Sleft + Swidth + 20;
0280 Sheight   = 20;
0281 Swidth    = 150;
0282 sth = uicontrol(panel, 'Style', 'edit', ...
0283     'String', 'obj', ...
0284     'FontSize', fontsize, ...
0285     'BackgroundColor', 'w', ...
0286     'HorizontalAlignment', 'left', ...
0287     'Position', [Sleft Sbottom Swidth Sheight], ...
0288     'Tag', 'objPrefixTxt');
0289
0290 %-- Append obj type
0291 pos = get(sth, 'Position');
0292 Sleft     = 10;
0293 Swidth    = 150;
0294 Sheight   = 20;
0295 Sbottom   = pos(2) - 20 - Sheight;
0296 cbh = uicontrol(panel, 'Style', 'checkbox', ...
0297     'String', 'Append object type', ...
0298     'FontSize', fontsize, ...
0299     'BackgroundColor', Gproperties.Gcol, ...
0300     'Value', 1, 'Position', [Sleft Sbottom Swidth Sheight], ...
0301     'Tag', 'appendObjTypeChk');
0302
0303 %-- Retrieve Btn
0304
0305 Bleft     = 10;
0306 Bheight   = 20;
0307 Bwidth    = 100;
0308 Bbottom   = Ppos(2) + 10;
0309 pbh = uicontrol(panel, 'Style', 'pushbutton', 'String', 'Retrieve', ...
0310     'Position', [Bleft Bbottom Bwidth Bheight ], ...
0311     'FontSize', fontsize, ...
0312     'Fontweight', 'bold', ...
0313     'Tag', 'retrieveBtn');

```

```

0314 set(pbh, 'Callback', {'cb_retrieveBtn', pbh, mainfig});
0315
0316
0317
0318 end
0319
0320 %-----
0321 % Create the Query panel
0322 %
0323 function panel = create_query_panel(panel, Gproperties)
0324
0325 Ppos      = get(panel, 'Position');
0326 mainfig = findobj('Tag', 'LTPDARepomainfig');
0327
0328 %-- Build conditions
0329 buildConditions(panel);
0330
0331 end
0332
0333 %-----
0334 % Create the submit panel
0335 %
0336 function panel = create_submit_panel(panel, Gproperties)
0337
0338 Ppos      = get(panel, 'Position');
0339 mainfig = findobj('Tag', 'LTPDARepomainfig');
0340
0341 %-- Text string
0342 Sleft     = 10;
0343 Sheight   = 20;
0344 Sbottom   = Ppos(2) + Ppos(4) - 20 - Gproperties.Gborder - Sheight;
0345 Swidth    = 200;
0346 fontsize  = getappdata(0, 'ltpda_repo_gui_fontsize');
0347 sth = uicontrol(panel, 'Style', 'text', ...
0348     'String', 'LTPDA Objects', ...
0349     'FontSize', fontsize, ...
0350     'BackgroundColor', Gproperties.Gcol, ...
0351     'HorizontalAlignment', 'center', ...
0352     'Position', [Sleft Sbottom Swidth Sheight]);
0353
0354 %-- Workspace list of LTPDA objects
0355 Lleft     = 10;
0356 Lheight   = 350;
0357 Lbottom   = Ppos(2) + Ppos(4) - 40 - Gproperties.Gborder - Lheight;
0358 Lwidth    = Swidth;
0359
0360 lbh = uicontrol(panel, 'Style', 'listbox', ...
0361     'String', {' '}, ...
0362     'Value', 1, ...
0363     'BackgroundColor', 'w', ...
0364     'FontSize', fontsize, ...

```

```

0365     'Max', 1000, ...
0366     'Position',[Lleft Lbottom Lwidth Lheight], ...
0367     'Tag', 'workspaceObjsList');
0368
0369 % Set callback
0370 set(lbh, 'Callback', {'cb_workspaceObjsList', lbh, mainfig});
0371
0372 % fill list
0373 objs = getWorkspaceObjs();
0374 setWorkspaceObjsList(objs);
0375
0376 %-- Refresh button
0377 pos      = get(lbh, 'Position');
0378 Bleft    = Lleft;
0379 Bheight  = 20;
0380 Bwidth   = 100;
0381 Bbottom  = pos(2) - Bheight - 10;
0382 pbh = uicontrol(panel, 'Style','pushbutton', 'String','Refresh list', ...
0383     'Position',[Bleft Bbottom Bwidth Bheight ], ...
0384     'Tag', 'refreshListBtn');
0385 set(pbh, 'Callback', {'cb_refreshObjsList', pbh, mainfig});
0386
0387
0388 %-- Info panel
0389 pos = get(lbh, 'Position');
0390 Sleft  = 10 + pos(1) + pos(3);
0391 Sheight = 150;
0392 Sbottom = Ppos(2) + Ppos(4) - 40 - Gproperties.Gborder - Sheight;
0393 Swidth  = 440;
0394 sth = uicontrol(panel, 'Style','text', ...
0395     'String','', ...
0396     'FontSize', fontsize, ...
0397     'BackgroundColor', 'w', ...
0398     'ForegroundColor', 'b', ...
0399     'HorizontalAlignment', 'left', ...
0400     'Position',[Sleft Sbottom Swidth Sheight], ...
0401     'Tag', 'ltpdaObjInfo');
0402
0403
0404 %-- experiment title
0405 pos      = get(sth, 'Position');
0406 Sheight  = 20;
0407 Swidth   = 150;
0408 Sbottom  = pos(2) - 20 - Sheight;
0409
0410 sth = uicontrol(panel, 'Style','text', ...
0411     'String','Experiment Title', ...
0412     'FontSize', fontsize, ...
0413     'BackgroundColor', Gproperties.Gcol, ...
0414     'HorizontalAlignment', 'left', ...
0415     'Position',[Sleft Sbottom Swidth Sheight]);

```

```

0416
0417 Tleft    = Sleft + Swidth + 20;
0418 Theight  = 20;
0419 Twidth   = 270;
0420 Tbottom  = Sbottom;
0421
0422 sth = uicontrol(panel, 'Style', 'edit', ...
0423     'String', '', ...
0424     'FontSize', fontsize, ...
0425     'BackgroundColor', 'w', ...
0426     'HorizontalAlignment', 'left', ...
0427     'Position', [Tleft Tbottom Twidth Theight], ...
0428     'Tag', 'experimentTitle');
0429
0430 %-- experiment description
0431 Sbottom = Sbottom - 20 - Theight;
0432
0433 sth = uicontrol(panel, 'Style', 'text', ...
0434     'String', 'Experiment description', ...
0435     'FontSize', fontsize, ...
0436     'BackgroundColor', Gproperties.Gcol, ...
0437     'HorizontalAlignment', 'left', ...
0438     'Position', [Sleft Sbottom Swidth Sheight]);
0439
0440 Extra = 60;
0441 Tbottom = Sbottom + 20 - Theight - Extra;
0442 Theight = Theight + Extra;
0443 sth = uicontrol(panel, 'Style', 'edit', ...
0444     'String', '', ...
0445     'FontSize', fontsize, ...
0446     'BackgroundColor', 'w', ...
0447     'Max', 100, ...
0448     'HorizontalAlignment', 'left', ...
0449     'Position', [Tleft Tbottom Twidth Theight], ...
0450     'Tag', 'experimentDescription');
0451
0452 %-- reference ids
0453 Sbottom = Sbottom - 20 - Theight;
0454
0455 sth = uicontrol(panel, 'Style', 'text', ...
0456     'String', 'Reference IDs', ...
0457     'FontSize', fontsize, ...
0458     'BackgroundColor', Gproperties.Gcol, ...
0459     'HorizontalAlignment', 'left', ...
0460     'Position', [Sleft Sbottom Swidth Sheight]);
0461
0462 Theight = 20;
0463 Tbottom = Sbottom + 20 - Theight;
0464 sth = uicontrol(panel, 'Style', 'edit', ...
0465     'String', '', ...
0466     'FontSize', fontsize, ...

```

```

0467     'BackgroundColor', 'w',...
0468     'Max', 1,...
0469     'HorizontalAlignment', 'left',...
0470     'Position',[Tleft Tbottom Twidth Theight],...
0471     'Tag', 'referenceIDs');
0472
0473 %-- additional comments
0474 Sbottom = Sbottom - 20 - Theight;
0475
0476 sth = uicontrol(panel,'Style','text',...
0477     'String','Additional Comments',...
0478     'FontSize', fontsize,...
0479     'BackgroundColor', Gproperties.Gcol,...
0480     'HorizontalAlignment', 'left',...
0481     'Position',[Sleft Sbottom Swidth Sheight]);
0482
0483 Extra = 30;
0484 Tbottom = Sbottom + 20 - Theight - Extra;
0485 Theight = Theight + Extra;
0486 sth = uicontrol(panel,'Style','edit',...
0487     'String','',...
0488     'FontSize', fontsize,...
0489     'BackgroundColor', 'w',...
0490     'Max', 100,...
0491     'HorizontalAlignment', 'left',...
0492     'Position',[Tleft Tbottom Twidth Theight],...
0493     'Tag', 'additionalComments');
0494
0495 %-- additional authors
0496 Sbottom = Sbottom - 20 - Theight;
0497
0498 sth = uicontrol(panel,'Style','text',...
0499     'String','Additional Authors',...
0500     'FontSize', fontsize,...
0501     'BackgroundColor', Gproperties.Gcol,...
0502     'HorizontalAlignment', 'left',...
0503     'Position',[Sleft Sbottom Swidth Sheight]);
0504
0505 Extra = 0;
0506 Tbottom = Sbottom + 20 - Theight - Extra;
0507 Theight = Theight + Extra;
0508 sth = uicontrol(panel,'Style','edit',...
0509     'String','',...
0510     'FontSize', fontsize,...
0511     'BackgroundColor', 'w',...
0512     'Max', 100,...
0513     'HorizontalAlignment', 'left',...
0514     'Position',[Tleft Tbottom Twidth Theight],...
0515     'Tag', 'additionalAuthors');
0516
0517

```



```

0518 %-- Submit Btn
0519
0520 pos      = get(lbh, 'Position');
0521 Bleft    = Lleft;
0522 Bheight  = 20;
0523 Bwidth   = 100;
0524 Bbottom  = Ppos(2) + 10;
0525 pbh = uicontrol(panel, 'Style', 'pushbutton', 'String', 'Submit', ...
0526     'Position', [Bleft Bbottom Bwidth Bheight ], ...
0527     'FontSize', fontsize, ...
0528     'Fontweight', 'bold', ...
0529     'Tag', 'submitBtn');
0530 set(pbh, 'Callback', {'cb_submitBtn', pbh, mainfig});
0531
0532
0533
0534 end
0535
0536
0537 %-----
0538 % Create the server connection panel
0539 %
0540 function panel = create_connection_panel(panel, Gproperties)
0541
0542
0543 % %-- Panel
0544 % Pleft    = Gproperties.Gborder/Gproperties.Gheight;
0545 % Pbottom  = Gproperties.Gborder/Gproperties.Gheight;
0546 % Pwidth   = 1-2*Gproperties.Gborder/Gproperties.Gwidth;
0547 % Pheight  = 0.4;
0548 %
0549 % hp = uipanel('Parent', mainfig, ...
0550 %             'Title', 'Server Connection', ...
0551 %             'FontSize', 14, ...
0552 %             'BackgroundColor', Gproperties.Gcol, ...
0553 %             'Position', [Pleft Pbottom Pwidth Pheight], ...
0554 %             'BorderType', 'etchedout', ...
0555 %             'HighlightColor', [0.1 0.9 0.1]);
0556
0557 Ppos      = get(panel, 'Position');
0558 mainfig = findobj('Tag', 'LTPDARepomainfig');
0559
0560 %-- Server selection
0561 Sleft     = 10;
0562 Sbottom   = Ppos(2) + Ppos(4) - 40-Gproperties.Gborder;
0563 Swidth    = 150;
0564 Sheight   = 20;
0565 fontsize  = getappdata(0, 'ltpda_repo_gui_fontsize');
0566
0567 pmh = uicontrol(panel, 'Style', 'popupmenu', ...
0568     'String', {'localhost', '130.75.117.67'}, ...

```



```

0569     'FontSize', fontsize, ...
0570     'BackgroundColor', Gproperties.Gcol, ...
0571     'Value', 1, 'Position', [Sleft Sbottom Swidth Sheight], ...
0572     'Tag', 'serverSelect');
0573 set(pmh, 'Callback', {'cb_serverSelect', pmh, mainfig});
0574
0575 %-- Server Txt entry
0576 Tleft = Sleft + Swidth + 10;
0577 Tbottom = Sbottom;
0578 Twidth = 200;
0579 Theight = 20;
0580
0581 eth = uicontrol(panel, 'Style', 'edit', ...
0582     'String', 'localhost', ...
0583     'FontSize', fontsize, ...
0584     'BackgroundColor', 'w', ...
0585     'Position', [Tleft Tbottom Twidth Theight], ...
0586     'Tag', 'serverTxt');
0587
0588 %-- Text string
0589 sth = uicontrol(panel, 'Style', 'text', ...
0590     'String', 'Server hostname', ...
0591     'FontSize', fontsize, ...
0592     'BackgroundColor', Gproperties.Gcol, ...
0593     'HorizontalAlignment', 'left', ...
0594     'Position', [Tleft+Twidth+10 Tbottom 120 Theight]);
0595
0596 %-- Database selection
0597 pmh = uicontrol(panel, 'Style', 'popupmenu', ...
0598     'String', {'ltpda', 'utp', 'test'}, ...
0599     'FontSize', fontsize, ...
0600     'BackgroundColor', Gproperties.Gcol, ...
0601     'Value', 1, 'Position', [Sleft Sbottom-Sheight-10 Swidth Sheight], ...
0602     'Tag', 'dbSelect');
0603 set(pmh, 'Callback', {'cb_dbSelect', pmh, mainfig});
0604
0605 %-- Database Txt entry
0606 eth = uicontrol(panel, 'Style', 'edit', ...
0607     'String', 'ltpda', ...
0608     'FontSize', fontsize, ...
0609     'BackgroundColor', 'w', ...
0610     'Position', [Tleft Tbottom-Theight-10 Twidth Theight], ...
0611     'Tag', 'databaseTxt');
0612
0613 %-- Text string
0614 sth = uicontrol(panel, 'Style', 'text', ...
0615     'String', 'Database', ...
0616     'FontSize', fontsize, ...
0617     'BackgroundColor', Gproperties.Gcol, ...
0618     'HorizontalAlignment', 'left', ...
0619     'Position', [Tleft+Twidth+10 Tbottom-Theight-10 120 Theight]);

```

```

0620
0621 %-- Available DBs button
0622 pos      = get(sth, 'Position');
0623 Bleft     = pos(1) + pos(3) + 20;
0624 Bbottom   = pos(2);
0625 Bwidth    = 100;
0626 Bheight   = 20;
0627 pbh = uicontrol(panel, 'Style', 'pushbutton', 'String', 'Get DBs', ...
0628     'Position', [Bleft Bbottom Bwidth Bheight ], ...
0629     'Tag', 'getDbBtn');
0630 set(pbh, 'Callback', {'cb_getDBs', pbh, mainfig});
0631
0632 %-- Login Btn
0633 pos      = get(pmh, 'Position');
0634 Bleft     = pos(1);
0635 Bbottom   = pos(2) - pos(4) - 20;
0636 Bwidth    = 100;
0637 Bheight   = 20;
0638
0639 pbh = uicontrol(panel, 'Style', 'pushbutton', 'String', 'Connect', ...
0640     'Position', [Bleft Bbottom Bwidth Bheight ], ...
0641     'Tag', 'connectBtn');
0642 set(pbh, 'Callback', {'cb_connectBtn', pbh, mainfig});
0643
0644 %-- Status Txt
0645 Sleft     = Bleft;
0646 Sbottom   = Ppos(2)+40;
0647 Swidth    = 60;
0648 Sheight   = 20;
0649
0650 sth = uicontrol(panel, 'Style', 'text', ...
0651     'String', 'Status:', ...
0652     'FontSize', fontsize, ...
0653     'BackgroundColor', Gproperties.Gcol, ...
0654     'HorizontalAlignment', 'left', ...
0655     'Position', [Sleft Sbottom Swidth Sheight]);
0656
0657 pos = get(sth, 'Position');
0658 Sleft = Sleft + Swidth + 20;
0659 Swidth = Ppos(3) - pos(2) - pos(3) - Gproperties.Gborder;
0660 Sheight = 60;
0661 Sbottom = Ppos(2) + 10;
0662 sth = uicontrol(panel, 'Style', 'text', ...
0663     'String', 'not connected', ...
0664     'FontSize', fontsize, ...
0665     'BackgroundColor', 'w', ...
0666     'HorizontalAlignment', 'left', ...
0667     'Max', 100, ...
0668     'Position', [Sleft Sbottom Swidth Sheight], ...
0669     'Tag', 'statusTxt');
0670

```

```
0671
0672 end
0673
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_HFT90D.m

win_HFT90D

PURPOSE

WIN_HFT90D return HFT90D window sample for input z.

SYNOPSIS

```
function w = win_HFT90D(z)
```

DESCRIPTION

WIN_HFT90D return HFT90D window sample for input z.

M Hewitson 18-05-07

\$Id: win_HFT90D.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_HFT90D(z)
0002
0003 % WIN_HFT90D return HFT90D window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT90D.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 z = z * 2 * pi;
0011 v = (1 - 1.942604 * cos (z) + ...
0012      1.340318 * cos (2 * z) - 0.440811 * cos (3 * z) + ...
0013      0.043097 * cos (4 * z));
0014
0015 % Make window struct
0016 w.name      = 'HFT90D';
0017 w.alpha     = 0;
0018 w.psll      = 90.2;
0019 w.rov       = 76;
0020 w.nenbw     = 3.8832;
0021 w.w3db      = 3.8320;
0022 w.flatness  = -0.0039;
0023 w.ws        = sum(v);
0024 w.ws2       = sum(v.*v);
0025 w.win       = v;
0026
0027 % END
```

[Home](#) > [classes](#) > [@miir](#) > char.m

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 1.4 2007/12/06 17:09:15 ingo 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.
- [eq](#) EQ overloads the == operator for miir objects.
- [get](#) GET get miir properties.
- [ne](#) NE overloads the ~= operator for miir objects.
- [redesign](#) REDESIGN redesign the input filter to work for the given sample rate.
- [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.
- **xmlparse** XMLPARSE creates a miir filter from a miir xml file.

SOURCE CODE

```

0001 function pstr = char(w, varargin)
0002 % CHAR convert a miir object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a miir object into a string.
0007 %
0008 % CALL:          string = char(miir)
0009 %
0010 % VERSION:       $Id: char.m,v 1.4 2007/12/06 17:09:15 ingo Exp $
0011 %
0012 % HISTORY:       04-02-2007 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION= '$Id: char.m,v 1.4 2007/12/06 17:09:15 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(w, 'miir') && ischar(varargin{1})
0022         in = char(varargin{1});
0023         if strcmp(in, 'Params')
0024             pstr = plist();
0025             return
0026         elseif strcmp(in, 'Version')
0027             pstr = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 pstr = [];
0034 pstr = [pstr w.name];
0035 pstr = [pstr ', fs=' num2str(w.fs)];
0036 pstr = [pstr ', ntaps=' num2str(w.ntaps)];
0037 pstr = [pstr ', a=' num2str(w.a)];
0038 pstr = [pstr ', b=' num2str(w.b)];
0039 pstr = [pstr ', gain=' num2str(w.gain)];
0040
0041 pstr = strrep(pstr, '_', '\_');
0042
0043 % END

```


[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_HFT95.m

win_HFT95

PURPOSE

WIN_HFT95 return HFT95 window sample for input z.

SYNOPSIS

function w = win_HFT95(z)

DESCRIPTION

WIN_HFT95 return HFT95 window sample for input z.

M Hewitson 18-05-07

\$Id: win_HFT95.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_HFT95(z)
0002
0003 % WIN_HFT95 return HFT95 window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT95.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 z = z * 2 * pi;
0011 v = (1 - 1.9383379 * cos (z) + ...
0012      1.3045202 * cos (2 * z) - 0.4028270 * cos (3 * z) + ...
0013      0.0350665 * cos (4 * z));
0014
0015 % Make window struct
0016 w.name      = 'HFT95';
0017 w.alpha     = 0;
0018 w.psll      = 95;
0019 w.rov       = 75.6;
0020 w.nenbw     = 3.8112;
0021 w.w3db      = 3.759;
0022 w.flatness  = 0.0044;
0023 w.ws        = sum(v);
0024 w.ws2       = sum(v.*v);
0025 w.win       = v;
0026
0027 % END
```

[Home](#) > [classes](#) > [@param](#) > char.m

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 1.5 2007/12/11 17:27:09 ingo 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
- [eq](#) EQ overloads the == operator for param objects.
- [get](#) GET get parameter properties.
- [mux](#) MUX concatonate params into a vector.
- [ne](#) NE overloads the ~= operator for param objects.
- [save](#) SAVE a param object to file.

- **set** SET set a parameter property.
- **subsasgn** SUBSASGN define index assignment for parameter properties.
- **subsref** SUBSREF Define field name indexing for parameter objects.
- **xmlparse** XMLPARSE creates a parameter from a parameter xml file.

SOURCE CODE

```

0001 function pstr = char(p, varargin)
0002 % CHAR convert a param object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a param object into a string.
0007 %
0008 % CALL:          string = char(p)
0009 %
0010 % VERSION:       $Id: char.m,v 1.5 2007/12/11 17:27:09 ingo Exp $
0011 %
0012 % HISTORY:       04-02-2007 M Hewitson
0013 %                  Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: char.m,v 1.5 2007/12/11 17:27:09 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(p, 'param') && ischar(varargin{1})
0022         in = char(varargin{1});
0023         if strcmp(in, 'Params')
0024             pstr = plist;
0025             return
0026         elseif strcmp(in, 'Version')
0027             pstr = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 if isnumeric(get(p, 'val'))
0034     pstr = [get(p, 'key') ' : ' ltpda_mat2str(get(p, 'val'))];
0035 else
0036     pstr = [get(p, 'key') ' : ' char(get(p, 'val'))];
0037 end
0038
0039
0040
0041 % END

```


[Home](#) > [classes](#) > [@fsdata](#) > reshapeF.m

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 1.4 2007/11/30 19:36:05 ingo Exp \$

HISTORY: 31-01-2007 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [fsdata](#) FSDATA frequency-series object class constructor.
- [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)
```

```

0002 % RESHAPEF reshape the frequency vector to match the y vector.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: RESHAPEF reshape the frequency vector to match the y vector.
0007 %
0008 % CALL:          fsd = reshapeF(fsd)
0009 %
0010 % INPUT:         fsd - frequency-series object
0011 %
0012 % OUTPUT        fsd - the reshaped frequency-series object
0013 %
0014 % VERSION:       $Id: reshapeF.m,v 1.4 2007/11/30 19:36:05 ingo Exp $
0015 %
0016 % HISTORY:       31-01-2007 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: reshapeF.m,v 1.4 2007/11/30 19:36:05 ingo Exp $';
0022
0023 if nargin == 2 && isa(fsd, 'fsdata') && ischar(varargin{1})
0024     if strcmp(varargin{1}, 'Params')
0025         fsd = plist();
0026         return
0027     elseif strcmp(varargin{1}, 'Version')
0028         fsd = VERSION;
0029         return
0030     end
0031 end
0032
0033 if size(fsd.x, 1) ~= size(fsd.y,1)
0034     fsd.x = [fsd.x].';
0035 end
0036 if size(fsd.x, 1) ~= size(fsd.y,1)
0037     error('### frequency and data vectors are not the same length.');
```

```

0038 end
0039
0040
0041 % END
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_Hamming.m

win_Hamming

PURPOSE

WIN_HAMMING return Hamming window sample for input z.

SYNOPSIS

function w = win_Hamming(z)

DESCRIPTION

WIN_HAMMING return Hamming window sample for input z.

M Hewitson 18-05-07

\$Id: win_Hamming.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_Hamming(z)
0002
0003 % WIN_HAMMING return Hamming window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Hamming.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```



```
0010 v = 0.54 - 0.46 * cos(2*pi*z);
0011
0012 % Make window struct
0013 w.name      = 'Hamming';
0014 w.alpha     = 0;
0015 w.psll      = 42.7;
0016 w.rov       = 50;
0017 w.nenbw     = 1.3628;
0018 w.w3db      = 1.3008;
0019 w.flatness  = -1.7514;
0020 w.ws        = sum(v);
0021 w.ws2       = sum(v.*v);
0022 w.win       = v;
0023
0024 % END
```

[Home](#) > [classes](#) > [@tsdata](#) > reshapeT.m

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 1.4 2007/11/30 19:36:05 ingo 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 %
```

```

0006 % DESCRIPTION: RESHAPET reshape the time-vector to match the x vector.
0007 %
0008 % CALL:          tsdata = reshape(tsdata)
0009 %
0010 % VERSION:       $Id: reshapeT.m,v 1.4 2007/11/30 19:36:05 ingo Exp $
0011 %
0012 % HISTORY:       31-01-2007 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: reshapeT.m,v 1.4 2007/11/30 19:36:05 ingo Exp $';
0018
0019 if nargin == 2 && isa(ts, 'tsdata') && ischar(varargin{1})
0020     if strcmp(varargin{1}, 'Params')
0021         ts = plist();
0022         return
0023     elseif strcmp(varargin{1}, 'Version')
0024         ts = VERSION;
0025         return
0026     end
0027 end
0028
0029 if size(ts.x, 1) ~= size(ts.y,1)
0030     ts.x = [ts.x].';
0031 end
0032
0033 % END

```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_Hanning.m

win_Hanning

PURPOSE

WIN_HANNING return Hanning window sample for input z.

SYNOPSIS

function w = win_Hanning(z)

DESCRIPTION

WIN_HANNING return Hanning window sample for input z.

M Hewitson 18-05-07

\$Id: win_Hanning.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_Hanning(z)
0002
0003 % WIN_HANNING return Hanning window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Hanning.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 v = 0.5 * (1 - cos(2*pi*z));
0011
0012 % Make window struct
0013 w.name      = 'Hanning';
0014 w.alpha     = 0;
0015 w.psll      = 31.5;
0016 w.rov       = 50;
0017 w.nenbw     = 1.5;
0018 w.w3db      = 1.4382;
0019 w.flatness  = -1.4236;
0020 w.ws        = sum(v);
0021 w.ws2       = sum(v.*v);
0022 w.win       = v;
0023
0024 % END
```

Generated on Tue 22-Jan-2008 10:39:13 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 1.4 2007/11/30 19:36:05 ingo Exp $

HISTORY:    31-01-07 M Hewitson
              Creation

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

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function xy = reshapeX(xy, varargin)
0002 % RESHAPEX reshape the X to match the Y vector.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: RESHAPEX reshape the X to match the Y vector.
0007 %
0008 % CALL:      xydata = reshape(xydata)
0009 %
```

```

0010 % VERSION:      $Id: reshapeX.m,v 1.4 2007/11/30 19:36:05 ingo Exp $
0011 %
0012 % HISTORY:       31-01-07 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: reshapeX.m,v 1.4 2007/11/30 19:36:05 ingo Exp $';
0018
0019 if nargin == 2 && isa(xy, 'xydata') && ischar(varargin{1})
0020     if strcmp(varargin{1}, 'Params')
0021         xy = plist();
0022         return
0023     elseif strcmp(varargin{1}, 'Version')
0024         xy = VERSION;
0025         return
0026     end
0027 end
0028
0029 if size(xy.x, 1) ~= size(xy.y,1)
0030     xy.x = [xy.x].';
0031 end
0032
0033 % END

```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_Nuttall3.m

win_Nuttall3

PURPOSE

WIN_NUTTALL3 return Nuttall3 window sample for input z.

SYNOPSIS

function w = win_Nuttall3(z)

DESCRIPTION

WIN_NUTTALL3 return Nuttall3 window sample for input z.

M Hewitson 18-05-07

\$Id: win_Nuttall3.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_Nuttall3(z)
0002
0003 % WIN_NUTTALL3 return Nuttall3 window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Nuttall3.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```



```
0010 z = z * 2 * pi;
0011 v = 0.375 - 0.5 * cos (z) + 0.125 * cos (2 * z);
0012
0013 % Make window struct
0014 w.name      = 'Nuttall3';
0015 w.alpha     = 0;
0016 w.psll      = 46.7;
0017 w.rov       = 64.7;
0018 w.nenbw     = 1.9444;
0019 w.w3db      = 1.8496;
0020 w.flatness  = -0.8630;
0021 w.ws        = sum(v);
0022 w.ws2       = sum(v.*v);
0023 w.win       = v;
0024
0025 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@provenance](#) > char.m

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 1.5 2008/01/07 10:21:53 ingo 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.
- [eq](#) EQ overloads the == operator for provenance objects.
- [find_real_ip](#) FIND_REAL_IP finds the real ip address and not the localhost address (127.0.0.1)
- [get](#) GET get provenance properties.
- [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.
- **subsasgn** SUBSASGN define index assignment for provenance properties.
- **subsref** SUBSREF Define field name indexing for provenance objects.
- **xmlparse** XMLPARSE creates a provenance from a provenance xml file.

SOURCE CODE

```

0001 function pstr = char(p, varargin)
0002 % CHAR convert a provenance object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a provenance object into a string.
0007 %
0008 % CALL:          string = char(prov)
0009 %
0010 % VERSION:       $Id: char.m,v 1.5 2008/01/07 10:21:53 ingo Exp $
0011 %
0012 % HISTORY:       04-02-2007 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: char.m,v 1.5 2008/01/07 10:21:53 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(p, 'provenance') && ischar(varargin{1})
0022         in = char(varargin{1});
0023         if strcmp(in, 'Params')
0024             pstr = plist;
0025             return
0026         elseif strcmp(in, 'Version')
0027             pstr = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 pstr = sprintf('created by %s@%s[%s] on %s/%s/%s at %s', ...
0034               get(p, 'creator'), ...
0035               get(p, 'hostname'), ...
0036               get(p, 'ip'), ...
0037               get(p, 'os'), ...
0038               get(p, 'matlab_version'), ...
0039               get(p, 'ltpda_version'), ...
0040               char(get(p, 'created')));
0041
0042
0043 % END

```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@mfir](#) > resp.m

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
 or
 f - a vector of frequency values

OUTPUTS: a - an analysis object

The response is calculated as:

$$H(s) = \text{gain} * (a_0 + a_1 \exp(s/fs) + \dots + a_n \exp((n-1)s/fs))$$

VERSION: \$Id: resp.m,v 1.7 2007/12/10 18:07:17 ingo Exp \$

HISTORY: 27-08-02 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- **set** SET set a mfir object property.

This function is called by:

SOURCE CODE

```

0001 function [varargout] = resp(varargin)
0002 % RESP Make a frequency response of the filter.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: RESP Make a frequency response of the filter.
0007 %               The input filter should be a mfir object.
0008 %
0009 %               The response is returned as a frequency-series in an analysis
0010 object.
0011 %
0012 %               If no outputs are specified, the frequency-series is plotted as
0013 mag/deg.
0014 % CALL:         a = resp(filt);
0015 %               a = resp(filt, pl);
0016 %               resp(filt, pl);
0017 %
0018 % PARAMETERS: f1      - the start frequency
0019 %              f2      - the stop frequency
0020 %              nf      - number of evaluation points
0021 %              or
0022 %              f        - a vector of frequency values
0023 %
0024 % OUTPUTS:      a      - an analysis object
0025 %
0026 % The response is calculated as:
0027 %
0028 %                $H(s) = \text{gain} * (a_0 + a_1 \exp(s/fs) + \dots + a_n \exp((n-1)s/fs))$ 
0029 %
0030 % VERSION: $Id: resp.m,v 1.7 2007/12/10 18:07:17 ingo Exp $
0031 %
0032 %
0033 % HISTORY: 27-08-02 M Hewitson
0034 %           Creation
0035 %
0036 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0037

```

```

0038 ALGONAME = mfilename;
0039 VERSION   = '$Id: resp.m,v 1.7 2007/12/10 18:07:17 ingo Exp $';
0040
0041 % 'Params' Call
0042 if nargin == 2
0043     if isa(varargin{1}, 'mfir') && strcmp(varargin{2}, 'Params')
0044         varargout{1} = plist();
0045         return
0046     elseif isa(varargin{1}, 'mfir') && strcmp(varargin{2}, 'Version')
0047         varargout{1} = VERSION;
0048         return
0049     end
0050 end
0051
0052 % get input filter
0053 filt = varargin{1};
0054 if ~isa(filt, 'mfir')
0055     error('### first input must be an mfir filter object.');
```

```

0056 end
0057
0058
0059 % create empty input if it is not specified
0060 if nargin == 1
0061     pl = plist();
0062 else
0063     pl = varargin{2};
0064 end
0065
0066 % create empty output parameter list
0067 plo = plist();
0068
0069 % parse input parameter list
0070 f1    = find(pl, 'f1');
0071 f2    = find(pl, 'f2');
0072 ndata = find(pl, 'nf');
0073 f     = find(pl, 'f');
0074
0075 % fill parameter list
0076 if isempty(f)
0077
0078     % we look for the other parameter set
0079     if isempty(f1)
0080         f1 = 0;
0081     end
0082     plo = append(plo, param('f1', f1));
0083     % we look for the other parameter set
0084     if isempty(f2)
0085         f2 = filt.fs/2;
0086     end
0087     plo = append(plo, param('f2', f2));
0088     % we look for the other parameter set
0089     if isempty(ndata)
```

```

0090     ndata = 1000;
0091     end
0092     plo = append(plo, param('nf', ndata));
0093
0094     % compute f from this
0095     f = linspace(f1, f2, ndata);
0096     plo = append(plo, param('f', f));
0097 else
0098     plo = append(plo, param('f', f));
0099
0100     ndata = length(f);
0101
0102 end
0103
0104 % append input filter to output parameter list
0105 plo = append(plo, param('filter', filt));
0106
0107 % compute Laplace vector
0108 s = -1i.*2*pi.*f;
0109
0110 % Compute filter response
0111 num = zeros(1, ndata);
0112 for n=1:length(filt.a)
0113     num = num + filt.a(n).*exp(s.*(n-1)/filt.fs);
0114 end
0115 dresp = num .* filt.gain; % apply the gain
0116
0117 % num = zeros(1, ndata);
0118 % for n=1:length(filt.a)
0119 %     num = num + filt.a(n).*exp(s.*(n-1)/filt.fs);
0120 % end
0121 % denom = zeros(1, ndata);
0122 % for n=1:length(filt.b)
0123 %     denom = denom + filt.b(n).*exp(s.*(n-1)/filt.fs);
0124 % end
0125 % dresp = num ./ denom;
0126 % dresp = dresp .* filt.g; % apply the gain
0127
0128 % mag = 20*log10(abs(dresp));
0129 % phase = angle(dresp)*180/pi;
0130
0131 %-----
0132 % Create an analysis object
0133
0134 % create new output fsdata
0135 fs = fsdata(f, dresp, filt.fs);
0136 fs = set(fs, 'name', sprintf('resp(%s)', filt.name));
0137
0138 % create new output history
0139 h = history(ALGONAME, VERSION, plo);
0140
0141 % make output analysis object

```



```
0142 b = ao(fs, h);
0143
0144 % set name
0145 b = setnh(b, 'name', sprintf('resp(%s)', filt.name));
0146
0147 % Outputs
0148 if nargout == 0
0149     plot(b)
0150 end
0151
0152 if nargout == 1
0153     varargout{1} = b;
0154 end
0155 if nargout > 1
0156     error('incorrect output arguments');
0157 end
0158
0159
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_Nuttall3a.m

win_Nuttall3a

PURPOSE

WIN_NUTTALL3A return Nuttall3a window sample for input z.

SYNOPSIS

function w = win_Nuttall3a(z)

DESCRIPTION

WIN_NUTTALL3A return Nuttall3a window sample for input z.

M Hewitson 18-05-07

\$Id: win_Nuttall3a.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_Nuttall3a(z)
0002
0003 % WIN_NUTTALL3A return Nuttall3a window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Nuttall3a.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 z = z * 2 * pi;
0011 v = 0.40897 - 0.5 * cos (z) + 0.09103 * cos (2 * z);
0012
0013 % Make window struct
0014 w.name      = 'Nuttall3a';
0015 w.alpha     = 0;
0016 w.psll      = 64.2;
0017 w.rov       = 61.2;
0018 w.nenbw     = 1.7721;
0019 w.w3db      = 1.6828;
0020 w.flatness  = -1.0453;
0021 w.ws        = sum(v);
0022 w.ws2       = sum(v.*v);
0023 w.win       = v;
0024
0025 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [mysql](#) > logindlg.m

logindlg

PURPOSE

LOGINDLG Dialog for visually secure login.

SYNOPSIS

```
function [varargout]=logindlg(varargin)
```

DESCRIPTION

LOGINDLG Dialog for visually secure login.

Examples:

```
[login password] = logindlg('Title','Login Title'); % Returns the login and  
password with the dialog title 'Login Title'.
```

```
password = logindlg; % Returns only the password with a default  
dialog title.
```

```
password = logindlg('Password','only'); % Displays only a password  
edit box and returns the password.
```

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)

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

```
0001 % LOGINDLG   Dialog for visually secure login.
0002 %   Examples:
0003 %       [login password] = logindlg('Title','Login Title'); % Returns
the login and
0004 %           password with the dialog title 'Login Title'.
0005 %
0006 %       password = logindlg; % Returns only the password with a default
0007 %           dialog title.
0008 %
0009 %       password = logindlg('Password','only'); % Displays only a password
0010 %           edit box and returns the password.
0011 %
0012 %
0013 %
0014 %
0015 % Author: Jeremy Smith
0016 % Date: September 24, 2005
0017 % Last Edit: July 7, 2006
0018 % Version: 1.2
0019 % Tested on: Matlab 7.0.4.365 (R14) Service Pack 2 and Matlab 7.1 SP 3
```

```
0020 % Description: custom login dialog because Matlab doesn't have an option
0021 %           for characters in an edit field to be replaced by asterixes
0022 %           (password security)
0023
0024 % Changelist:
0025 %   1.2: -Tab no longer triggers the OK button in the password box
0026 %       -Improved the script help
0027 %       -Removed horizontal alignment from buttons
0028 %       -Added the option to display only the password box
0029 %   1.1: -Added positioning code so it'll display in the center of the screen
0030 %       -If only one output is specified the password will be returned
0031 %           instead of the login as in Version 1.0
0032 %       -Escape will not only close the dialog if neither edit box is active
0033 %       -When the dialog appears the first edit box will be active
0034 %       -Added a few more comments
0035 %       -Removed the clc, it was left in by mistake in Version 1.0
0036
0037 function [varargout]=logindlg(varargin)
0038
0039 % Number of inputs check
0040 if nargin == 0 || nargin == 2 || nargin == 4
0041 else
0042     error('Incorrect number of input arguments.')
0043 end
0044
0045 % Input Type Check
0046 if ~isempty(find(cellfun(@ischar,varargin) == 0))
0047     error('Inputs must be strings.')
0048 end
0049
0050 % Title Option
0051 if nargin == 0
0052     Title = 'Login';
0053 elseif nargin == 2 && ~isempty(strmatch('title',lower(varargin)))
0054     Title = varargin{2};
0055 elseif nargin == 2 && isempty(strmatch('title',lower(varargin)))
0056     Title = 'Login';
0057 elseif nargin == 4 && ~isempty(strmatch('title',lower(varargin)))
0058     S = strmatch('title',lower(varargin));
0059     if S == 1
```

```
0060         Title = varargin{2};
0061     elseif S == 3
0062         Title = varargin{4};
0063     else
0064         error('Invalid title.')
0065     end
0066 else
0067     error('Invalid title.')
0068 end
0069
0070 % Password Option
0071 if nargin == 0
0072     Pass = 0;
0073 elseif nargin == 2 && ~isempty(strmatch('password', lower(varargin{1})))
0074     && ~isempty(strmatch('only', lower(varargin{2})))
0075     Pass = 1;
0076 elseif nargin == 4 && ~isempty(strmatch('password', lower(varargin))) &&
0077     ~isempty(strmatch('only', lower(varargin)))
0078     P = strmatch('password', lower(varargin));
0079     O = strmatch('only', lower(varargin));
0080     if P == 1 && O == 2
0081         Pass = 1;
0082     elseif P == 3 && O == 4
0083         Pass = 1;
0084     end
0085 elseif nargin == 2 && isempty(strmatch('password', lower(varargin))) == 1
0086     Pass = 0;
0087 else
0088     error('Invalid password option.')
0089 end
0090
0091 % Output Error Check
0092 if nargsout > 1 && Pass == 1 || nargsout > 2
0093     error('Too many output arguments.')
0094 end
0095
0096 % Get Properties
0097 Color = [240 240 240]/255; %get(0, 'DefaultUicontrolBackgroundColor');
0098
0099 % Determine the size and position of the login interface
0100 if Pass == 0
```

```

0099     Height = 9.5;
0100 else
0101     Height = 5.5;
0102 end
0103 set(0, 'Units', 'characters')
0104 Screen = get(0, 'screensize');
0105 Position = [Screen(3)/2-17.5 Screen(4)/2-4.75 35 Height];
0106 set(0, 'Units', 'pixels')
0107
0108 % Create the GUI
0109 gui.main = dialog('HandleVisibility','on',...
0110     'IntegerHandle','off',...
0111     'Menubar','none',...
0112     'NumberTitle','off',...
0113     'Name','Login',...
0114     'Tag','logindlg',...
0115     'Color',Color,...
0116     'Units','characters',...
0117     'Userdata','logindlg',...
0118     'Position',Position);
0119
0120 % Set the title
0121 if ischar(Title) == 1
0122     set(gui.main, 'Name', Title, 'Closerequestfcn', {@Cancel,
gui.main}, 'KeyPressfcn', {@Escape, gui.main})
0123 end
0124
0125 FONTSIZE = 8;
0126
0127 % Texts
0128 if Pass == 0
0129     gui.login_text = uicontrol(gui.
main, 'Style', 'text', 'BackgroundColor', Color,
'FontSize',
FONTSIZE, 'HorizontalAlign', 'left', 'Units', 'characters', 'String', 'Login', 'Position',
[1 7.65 20 1]);
0130 end
0131 gui.password_text = uicontrol(gui.main, 'Style', 'text', 'BackgroundColor',
Color,
'FontSize',

```



```

    FONTSIZE, 'HorizontalAlign', 'left', 'Units', 'characters', 'String', 'Password', 'Position',
    [1 4.15 20 1]);
0132
0133 % Edits
0134 if Pass == 0
0135     gui.edit1 = uicontrol(gui.
main, 'Style', 'edit', 'FontSize',
    FONTSIZE, 'HorizontalAlign', 'left', 'BackgroundColor', 'white', 'Units', 'characters', 'String', '', 'Position',
    [1 6.02 33 1.7]);
0136 end
0137 gui.edit2 = uicontrol(gui.
main, 'Style', 'edit', 'FontSize',
    FONTSIZE, 'HorizontalAlign', 'left', 'BackgroundColor', 'white', 'Units', 'characters', 'String', '', 'Position',
    [1 2.52 33 1.7], 'KeyPressFcn', {@KeyPress_Function, gui.main}, 'Userdata', '');
0138
0139 % Buttons
0140 gui.OK = uicontrol(gui.main, 'Style', 'push', 'BackgroundColor', Color,
    'FontSize', FONTSIZE, 'Units', 'characters', 'String', 'OK', 'Position', [12 .2
10 1.7], 'Callback', {@OK, gui.main});
0141 gui.Cancel = uicontrol(gui.main, 'Style', 'push', 'BackgroundColor',
    Color, 'FontSize', FONTSIZE, 'Units', 'characters', 'String', 'Cancel', 'Position',
    [23 .2 10 1.7], 'Callback', {@Cancel, gui.main});
0142
0143 setappdata(0, 'logindlg', gui) % Save handle data
0144 setappdata(gui.main, 'Check', 0) % Error check setup. If Check remains 0 an
empty cell array will be returned
0145
0146 if Pass == 0
0147     uicontrol(gui.edit1) % Make the first edit box active
0148 else
0149     uicontrol(gui.edit2) % Make the second edit box active if the first
isn't present
0150 end
0151
0152 % Pause the GUI and wait for a button to be pressed
0153 uiwait(gui.main)
0154
0155 Check = getappdata(gui.main, 'Check'); % Check to see if a button was pressed
0156
0157 % Format output

```

```

0158 if Check == 1
0159     if Pass == 0
0160         Login = get(gui.edit1, 'String');
0161     end
0162     Password = get(gui.edit2, 'Userdata');
0163
0164     if nargout == 1 % If only one output specified output Password
0165         varargout(1) = {Password};
0166     elseif nargout == 2 % If two outputs specified output both Login
and Password
0167         varargout(1) = {Login};
0168         varargout(2) = {Password};
0169     end
0170 else % If OK wasn't pressed output nothing
0171     if nargout == 1
0172         varargout(1) = {[]};
0173     elseif nargout == 2
0174         varargout(1) = {[]};
0175         varargout(2) = {[]};
0176     end
0177 end
0178
0179 delete(gui.main) % Close the GUI
0180 setappdata(0, 'logindlg', []) % Erase handles from memory
0181
0182 %% Hide Password
0183 function KeyPress_Function(h,eventdata,fig)
0184 % Function to replace all characters in the password edit box with
0185 % asterixes
0186 password = get(h, 'Userdata');
0187 key = get(fig, 'currentkey');
0188
0189 switch key
0190     case 'backspace'
0191         password = password(1:end-1); % Delete the last character in
the password
0192     case 'return' % This cannot be done through callback without making tab
to the same thing
0193         gui = getappdata(0, 'logindlg');
0194         OK([], [], gui.main);

```

```
0195     case 'tab' % Avoid tab triggering the OK button
0196         gui = getappdata(0, 'logindlg');
0197         uicontrol(gui.OK);
0198     otherwise
0199         password = [password get(fig, 'currentcharacter')]; % Add the
typed character to the password
0200 end
0201
0202 SizePass = size(password); % Find the number of asterixes
0203 if SizePass(2) > 0
0204     asterix(1,1:SizePass(2)) = '*'; % Create a string of asterixes the
same size as the password
0205     set(h, 'String', asterix) % Set the text in the password edit box to
the asterix string
0206 else
0207     set(h, 'String', '')
0208 end
0209
0210 set(h, 'Userdata', password) % Store the password in its current state
0211
0212 %% Cancel
0213 function Cancel(h,eventdata,fig)
0214 uiresume(fig)
0215
0216 %% OK
0217 function OK(h,eventdata,fig)
0218 % Set the check and resume
0219 setappdata(fig, 'Check', 1)
0220 uiresume(fig)
0221
0222 %% Escape
0223 function Escape(h,eventdata,fig)
0224 % Close the login if the escape button is pushed and neither edit box is
0225 % active
0226 key = get(fig, 'currentkey');
0227
0228 if isempty(strfind(key, 'escape')) == 0 && h == fig
0229     Cancel([], [], fig)
0230 end
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@miir](#) > resp.m

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
 or
 f - a vector of frequency values

OUTPUTS:
 a - an analysis object

The response is calculated as:

$$H(s) = \text{gain} * \frac{a_0 + a_1 \exp(s/fs) + \dots + a_n \exp((n-1)s/fs)}{b_0 + b_1 \exp(s/fs) + \dots + b_m \exp((m-1)s/fs)}$$

VERSION: \$Id: resp.m,v 1.5 2007/12/10 18:07:17 ingo Exp \$

NOTE: Some of the code below is taken from Mathworks's treeplot.m

HISTORY: 27-08-02 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [set](#) SET set a miir object property.

This function is called by:

SOURCE CODE

```

0001 function [varargout] = resp(varargin)
0002 % RESP Make a frequency response of the filter.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: RESP Make a frequency response of the filter.
0007 %               The input filter should be an miir object.
0008 %
0009 %               The response is returned as a frequency-series in an
0010 %               analysis object.
0011 %
0012 %               If no outputs are specified, the frequency-series is plotted as
0013 %               mag/deg.
0014 %
0015 % CALL:         a = resp(filt, pl)
0016 %               resp(filt, pl);
0017 %
0018 % PARAMETERS:  f1      - the start frequency
0019 %               f2      - the stop frequency
0020 %               nf      - number of evaluation points
0021 %               or
0022 %               f        - a vector of frequency values
0023 %
0024 % OUTPUTS:
0025 %               a        - an analysis object
0026 %
0027 % The response is calculated as:
0028 %
0029 %               a0 + a1*exp(s/fs) + ... + an*exp((n-1)s/fs)
0030 %   H(s) = gain * -----
0031 %               b0 + b1*exp(s/fs) + ... + bm*exp((m-1)s/fs)
0032 %
0033 % VERSION: $Id: resp.m,v 1.5 2007/12/10 18:07:17 ingo Exp $
0034 %

```

```

0035 % NOTE:      Some of the code below is taken from Mathworks's treeplot.m
0036 %
0037 % HISTORY: 27-08-02 M Hewitson
0038 %           Creation
0039 %
0040 %%%%%%%%%%%
0041
0042 ALGONAME = mfilename;
0043 VERSION  = '$Id: resp.m,v 1.5 2007/12/10 18:07:17 ingo Exp $';
0044
0045 % 'Params' Call
0046 if nargin == 2
0047     if isa(varargin{1}, 'miir') && strcmp(varargin{2}, 'Params')
0048         varargout{1} = plist();
0049         return
0050     elseif isa(varargin{1}, 'miir') && strcmp(varargin{2}, 'Version')
0051         varargout{1} = VERSION;
0052         return
0053     end
0054 end
0055
0056 % get input filter
0057 filt = varargin{1};
0058 if ~isa(filt, 'miir')
0059     error('### first input must be an miir filter object.');
```

```

0060 end
0061
0062
0063 % create empty input if it is not specified
0064 if nargin == 1
0065     pl = plist();
0066 else
0067     pl = varargin{2};
0068 end
0069
0070 % create empty output parameter list
0071 plo = plist();
0072
0073 % parse input parameter list
0074 f1    = find(pl, 'f1');
0075 f2    = find(pl, 'f2');
0076 ndata = find(pl, 'nf');
0077 f     = find(pl, 'f');
```

```

0078
0079 % fill parameter list
0080 if isempty(f)
0081
0082     % we look for the other parameter set
0083     if isempty(f1)
0084         f1 = 0;
0085     end
0086     plo = append(plo, param('f1', f1));

```

```

0087 % we look for the other parameter set
0088 if isempty(f2)
0089     f2 = filt.fs/2;
0090 end
0091 plo = append(plo, param('f2', f2));
0092 % we look for the other parameter set
0093 if isempty(ndata)
0094     ndata = 1000;
0095 end
0096 plo = append(plo, param('nf', ndata));
0097
0098 % compute f from this
0099 f = linspace(f1, f2, ndata);
0100 plo = append(plo, param('f', f));
0101 else
0102     plo = append(plo, param('f', f));
0103
0104     ndata = length(f);
0105
0106 end
0107
0108 % append input filter to output parameter list
0109 plo = append(plo, param('filter', filt));
0110
0111 % compute Laplace vector
0112 s = -1i.*2*pi.*f;
0113
0114 % Compute filter response
0115 num = zeros(1, ndata);
0116 for n=1:length(filt.a)
0117     num = num + filt.a(n).*exp(s.*(n-1)/filt.fs);
0118 end
0119 denom = zeros(1, ndata);
0120 for n=1:length(filt.b)
0121     denom = denom + filt.b(n).*exp(s.*(n-1)/filt.fs);
0122 end
0123 dresp = num ./ denom;
0124 dresp = dresp .* filt.gain; % apply the gain
0125
0126 % mag    = 20*log10(abs(dresp));
0127 % phase = angle(dresp)*180/pi;
0128
0129 %-----
0130 % Create an analysis object
0131
0132 % create new output fsdata
0133 fs = fsdata(f, dresp, filt.fs);
0134 fs = set(fs, 'name', sprintf('resp(%s)', filt.name));
0135
0136 % create new output history
0137 h = history(ALGONAME, VERSION, plo);
0138

```



```
0139 % make output analysis object
0140 b = ao(fs, h);
0141
0142 % set name
0143 b = setnh(b, 'name', sprintf('resp(%s)', filt.name));
0144
0145 % Outputs
0146 if nargout == 0
0147     plot(b)
0148 end
0149
0150 if nargout == 1
0151     varargout{1} = b;
0152 end
0153 if nargout > 1
0154     error('incorrect output arguments');
0155 end
0156
0157
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_Nuttall3b.m

win_Nuttall3b

PURPOSE

WIN_NUTTALL3B return Nuttall3b window sample for input z.

SYNOPSIS

```
function w = win_Nuttall3b(z)
```

DESCRIPTION

WIN_NUTTALL3B return Nuttall3b window sample for input z.

M Hewitson 18-05-07

\$Id: win_Nuttall3b.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_Nuttall3b(z)
0002
0003 % WIN_NUTTALL3B return Nuttall3b window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Nuttall3b.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 z = z * 2 * pi;
0011 v = 0.4243801 - 0.4973406 * cos (z) + 0.0782793 * cos (2 * z);
0012
0013 % Make window struct
0014 w.name      = 'Nuttall3b';
0015 w.alpha     = 0;
0016 w.psll      = 71.5;
0017 w.rov       = 59.8;
0018 w.nenbw     = 1.7037;
0019 w.w3db      = 1.6162;
0020 w.flatness  = -1.1352;
0021 w.ws        = sum(v);
0022 w.ws2       = sum(v.*v);
0023 w.win       = v;
0024
0025 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_Nuttall4.m

win_Nuttall4

PURPOSE

WIN_NUTTALL4 return Nuttall4 window sample for input z.

SYNOPSIS

function w = win_Nuttall4(z)

DESCRIPTION

WIN_NUTTALL4 return Nuttall4 window sample for input z.

M Hewitson 18-05-07

\$Id: win_Nuttall4.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_Nuttall4(z)
0002
0003 % WIN_NUTTALL4 return Nuttall4 window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Nuttall4.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.3125 - 0.46875 * cos(z) + 0.1875 * cos (2 * z) - 0.03125 * cos (3 * z);
0012
0013 % Make window struct
0014 w.name      = 'Nuttall4';
0015 w.alpha     = 0;
0016 w.psll      = 60.9;
0017 w.rov       = 70.5;
```

```
0018 w.nenbw      = 2.31;  
0019 w.w3db       = 2.1884;  
0020 w.flatness   = -0.6184;  
0021 w.ws         = sum(v);  
0022 w.ws2        = sum(v.*v);  
0023 w.win        = v;  
0024  
0025 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@time](#) > char.m

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 1.4 2008/01/14 20:03:52 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:

- [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.
- [ne](#) NE overloads the ~= 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.
- [xmladd](#) XMLADD add a time object to the given node of the input XML DOM
- [xmlparse](#) XMLPARSE creates a time from a time xml file.

SOURCE CODE

```

0001 function pstr = char(tt, varargin)
0002 % CHAR convert a time object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a time object into a string.
0007 %
0008 % CALL:          string = char(time)
0009 %
0010 % VERSION:       $Id: char.m,v 1.4 2008/01/14 20:03:52 ingo Exp $
0011 %
0012 % HISTORY:       23-07-2007 Diepholz
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: char.m,v 1.4 2008/01/14 20:03:52 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(tt, 'time') && ischar(varargin{1})
0022         in = char(varargin{1});
0023         if strcmp(in, 'Params')
0024             pstr = plist;
0025             return
0026         elseif strcmp(in, 'Version')
0027             pstr = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 pstr = tt.time_str;
0034
0035 % END

```

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

```
0001 function ltpda_allgrid(state)
0002 % LTPDA_ALLGRID Set all the grids to 'state'
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_ALLGRID Set all the grids to 'state'
0007 %
0008 % CALL:      ltpda_allgrid('state')
0009 %
```



```

0010 % VERSION:      $Id: ltpda_allgrid.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
0011 %
0012 % HISTORY:      17-11-2002 M Hewitson
0013 %                Creation.
0014 %
0015 %%%%%%%%%%%%%%%
0016
0017 if (strcmp(state, 'on') || strcmp(state, 'off'))
0018
0019     c = get(gcf, 'children');
0020
0021     for k=1:length(c)
0022
0023         t = get(c(k), 'Tag');
0024         if t=='
0025             set(c(k), 'XGrid', state);
0026             set(c(k), 'YGrid', state);
0027         end
0028
0029     end
0030 else
0031     error('state must be on or off');
0032 end
0033
0034 %
0035 % END

```

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.

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'

VERSION: \$Id: resp.m,v 1.5 2008/01/11 14:04:43 ingo Exp \$

HISTORY: 03-04-2007 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:

SOURCE CODE

```

0001 function [f,r] = resp(varargin)
0002 % RESP returns the complex response of the pole object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: RESP returns the complex response of the pole object.
0007 %
0008 % CALL:          [f,r] = resp(p, f);           % compute response for vector f
0009 %               [f,r] = resp(p, f1, f2, nf); % compute response from f1 to f2 in
nf
0010 %                                   steps.
0011 %               [f,r] = resp(p, pl);          % compute response from parameter
list.
0012 %
0013 % REMARK:        This is just a helper function. This function should only be
0014 %               called from class functions.
0015 %
0016 % PARAMETERS:    'f'          - a vector of frequencies to evaluate at
0017 %               or
0018 %               'f1'         - start frequency
0019 %               'f2'         - stop frequency
0020 %               'nf'         - number of evaluation frequencies
0021 %               'scale'      - spacing of frequencies: 'lin' or 'log'
0022 %
0023 % VERSION:       $Id: resp.m,v 1.5 2008/01/11 14:04:43 ingo Exp $
0024 %
0025 % HISTORY:       03-04-2007 M Hewitson
0026 %               Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 VERSION = '$Id: resp.m,v 1.5 2008/01/11 14:04:43 ingo Exp $';
0031
0032 % Check if this is a call for parameters or for the cvs-version number
0033 if nargin == 2
0034     if isa(varargin{1}, 'pole') && ischar(varargin{2})
0035         in = char(varargin{2});
0036         if strcmp(in, 'Params')
0037             f = plist();
0038             return
0039         elseif strcmp(in, 'Version')
0040             f = VERSION;
0041             return
0042         end
0043     end
0044 end
0045

```

```

0046 if nargin < 1
0047     error('### incorrect number of inputs.')
0048 end
0049
0050 % look at inputs
0051 p = varargin{1};
0052 if ~isa(p, 'pole')
0053     error('### first argument should be a pole object.');
```

```

0054 end
0055 % Now look at the pole
0056 f0 = get(p, 'f');
0057 Q = get(p, 'q');
```

```

0058
0059 % check inputs
0060 if nargin == 1
0061
0062     f = linspace(f0/10, 10*f0, 1000);
0063
0064 elseif nargin == 2
0065     if isa(varargin{2}, 'plist')
0066         pl = varargin{2};
0067         f = find(pl, 'f');
0068         f1 = find(pl, 'f1');
0069         f2 = find(pl, 'f2');
0070         nf = find(pl, 'nf');
0071         scale = find(pl, 'scale');
```

```

0072
0073         if ~isempty(f)
0074             % nothing
0075         elseif ~isempty(f1) && ~isempty(f2) && ~isempty(nf)
0076             if isempty(scale) || strcmp(scale, 'lin')
0077                 f = linspace(f1, f2, nf);
0078             elseif strcmp(scale, 'log')
0079                 f = logspace(log10(f1), log10(f2), nf);
0080             else
0081                 error('### unknown scale option for pole response.');
```

```

0082             end
0083         else
0084             error('### unknown parameter list for pole response.');
```

```

0085         end
0086
0087     else
0088         f = varargin{2};
0089     end
0090 elseif nargin == 4
0091     f1 = varargin{2};
0092     f2 = varargin{3};
0093     nf = varargin{4};
0094     f = linspace(f1, f2, nf);
0095 else
0096     error('### incorrect number of inputs.');
```

```

0097 end
0098

```

```
0099
0100 if Q>0.5
0101     re = 1 - (f.^2./f0^2);
0102     im = f ./ (f0*Q);
0103     r = 1./complex(re, im);
0104 else
0105     re = 1;
0106     im = f./f0;
0107     r = 1./complex(re, im);
0108 end
0109
0110
0111 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_Nuttall4a.m

win_Nuttall4a

PURPOSE [^](#)

WIN_NUTTALL4A return Nuttall4a window sample for input z.

SYNOPSIS [^](#)

function w = win_Nuttall4a(z)

DESCRIPTION [^](#)

WIN_NUTTALL4A return Nuttall4a window sample for input z.

M Hewitson 18-05-07

\$Id: win_Nuttall4a.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^](#)

This function calls:

This function is called by:

SOURCE CODE [^](#)

```
0001 function w = win_Nuttall4a(z)
0002
0003 % WIN_NUTTALL4A return Nuttall4a window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Nuttall4a.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.338946 - 0.481973 * cos (z) + 0.161054 * cos (2 * z) - 0.018027 * cos (3
* z);
0012
0013 % Make window struct
0014 w.name      = 'Nuttall4a';
0015 w.alpha     = 0;
0016 w.psll      = 82.6;
0017 w.rov       = 68;
```

```
0018 w.nenbw      = 2.1253;  
0019 w.w3db       = 2.0123;  
0020 w.flatness   = -0.7321;  
0021 w.ws         = sum(v);  
0022 w.ws2        = sum(v.*v);  
0023 w.win        = v;  
0024  
0025 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@timeformat](#) > char.m

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 1.2 2008/01/15 17:36:36 ingo 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.
- [eq](#) EQ overloads the == operator for timeformat objects.
- [ne](#) NE overloads the ~= operator for timeformat objects.
- [save](#) SAVE a timeformat object to file.
- [set](#) SET set timeformat properties.
- [subsasgn](#) SUBSASGN define index assignment for timeformat properties.
- [subsref](#) SUBSREF Define field name indexing for timeformat objects.
- [timeformat](#) TIMEFORMAT timeformat object class constructor.

- [xmlparse](#) XMLPARSE creates a timeformat from a timeformat xml file.

SOURCE CODE

```

0001 function pstr = char(tf, varargin)
0002 % CHAR convert a timeformat object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a timeformat object into a string.
0007 %
0008 % CALL:          string = char(timeformat)
0009 %
0010 % VERSION:       $Id: char.m,v 1.2 2008/01/15 17:36:36 ingo Exp $
0011 %
0012 % HISTORY:       23-07-2007 Diepholz
0013 %                  Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: char.m,v 1.2 2008/01/15 17:36:36 ingo Exp $';
0018
0019 % Check if this is a call for parameters or for the cvs-version number
0020 if nargin == 2
0021     if isa(tf, 'timeformat') && ischar(varargin{1})
0022         in = char(varargin{1});
0023         if strcmp(in, 'Params')
0024             pstr = plist();
0025             return
0026         elseif strcmp(in, 'Version')
0027             pstr = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 if ~isempty(tf.format_str)
0034     pstr = tf.format_str;
0035 else
0036     pstr = num2str(tf.format_nr);
0037 end
0038
0039 % END

```

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

```
0001 function ltpda_allxaxis(x1, x2)
0002 % LTPDA_ALLXSCALE Set all the x scales on the current figure.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_ALLXAXIS Set all the xaxis ranges on the current figure.
0007 %
0008 % CALL:      ltpda_allxaxis(x1, x2)
0009 %
```

```
0010 % VERSION:      $Id: ltpda_allxaxis.m,v 1.6 2007/07/30 16:04:09 ingo Exp $
0011 %
0012 % HISTORY:       17-11-2002 M Hewitson
0013 %                Creation.
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 h = findobj(gcf, 'Type', 'axes', '-not', 'Tag', 'legend');
0018
0019 set(h, 'XLim', [x1 x2])
0020
0021 %
0022 % END
```

[Home](#) > [classes](#) > [@pzmodel](#) > resp.m

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(p, f);           % compute response for vector f
a = resp(p, f1, f2, nf); % compute response from f1 to f2 in nf
                        steps.
a = resp(p, pl);          % compute response from parameter list.

```

% 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'

VERSION: \$Id: resp.m,v 1.9 2008/01/11 14:00:55 ingo Exp \$

HISTORY: 03-04-2007 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.
- [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.
- **set** SET sets a pole/zero model property.

This function is called by:

- **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

```

0001 function varargout = resp(varargin)
0002 % RESP returns the complex response of a pzmodel as an Analysis Object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: RESP returns the complex response of a pzmodel as an Analysis
0007 %               Object.
0008 %
0009 % CALL:         a = resp(p, f);           % compute response for vector f
0010 %               a = resp(p, f1, f2, nf); % compute response from f1 to f2 in nf
0011 %               steps.
0012 %               a = resp(p, pl);          % compute response from parameter list.
0013 %
0014 % % PARAMETERS: 'f' - a vector of frequencies to evaluate at
0015 %               or
0016 %               'f1' - start frequency
0017 %               'f2' - stop frequency
0018 %               'nf' - number of evaluation frequencies
0019 %               'scale' - spacing of frequencies: 'lin' or 'log'
0020 %
0021 % VERSION:      $Id: resp.m,v 1.9 2008/01/11 14:00:55 ingo Exp $
0022 %
0023 % HISTORY:      03-04-2007 M Hewitson
0024 %               Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0027
0028 ALGONAME = mfilename;
0029 VERSION  = '$Id: resp.m,v 1.9 2008/01/11 14:00:55 ingo Exp $';
0030
0031 % Check if this is a call for parameters
0032 if nargin == 2
0033     if isa(varargin{1}, 'pzmodel') && ischar(varargin{2})
0034         in = char(varargin{2});
0035         if strcmp(in, 'Params')
0036             varargout{1} = plist();
0037             return
0038         elseif strcmp(in, 'Version')
0039             varargout{1} = VERSION;

```

```

0040     return
0041 end
0042 end
0043 end
0044
0045 if nargin < 1
0046     error('### incorrect number of inputs.')
0047 end
0048
0049 % look at inputs
0050 pzm = varargin{1};
0051 if ~isa(pzm, 'pzmodel')
0052     error('### first argument should be a pzmodel object.');
```

```

0053 end
0054 % Now look at the model
0055 name = get(pzm, 'name');
0056 gain = get(pzm, 'gain');
0057 poles = get(pzm, 'poles');
0058 zeros = get(pzm, 'zeros');
0059 np = length(poles);
0060 nz = length(zeros);
0061
0062 % check inputs
0063 pl = [];
0064 scale = [];
0065
0066 if nargin == 1
0067
0068     f1 = getlowerFreq(pzm)/10;
0069     f2 = getupperFreq(pzm)*10;
0070     nf = 10000;
0071     f = linspace(f1, f2, nf);
0072     scale = 'lin';
0073
0074 elseif nargin == 2
0075     if isa(varargin{2}, 'plist')
0076         pl = varargin{2};
0077         f = find(pl, 'f');
0078         f1 = find(pl, 'f1');
0079         f2 = find(pl, 'f2');
0080         nf = find(pl, 'nf');
0081         scale = find(pl, 'scale');
```

```

0082
0083         if ~isempty(f)
0084             % nothing
0085         else
0086             if isempty(f1)
0087                 f1 = getlowerFreq(pzm)/10;
0088             end
0089             if isempty(f2)
0090                 f2 = getupperFreq(pzm)*10;

```

```

0091     end
0092     if isempty(nf)
0093         nf = 10000;
0094     end
0095     if isempty(scale)
0096         scale = 'lin';
0097     end
0098
0099     switch scale
0100     case {'lin', 'linear'}
0101         f = linspace(f1, f2, nf);
0102     case 'log'
0103         f = logspace(log10(f1), log10(f2), nf);
0104     end
0105
0106
0107 end
0108 %     if ~isempty(f1) && ~isempty(f2) && ~isempty(nf)
0109 %         if isempty(scale) || strcmp(scale, 'lin')
0110 %             elseif strcmp(scale, 'log')
0111 %             else
0112 %                 error('### unknown scale option for pzmodel response.');
```

```

0113 %             end
0114 %         else
0115 %             % build default
0116 %             f1 = getlowerFreq(pzm)/10;
0117 %             f2 = getupperFreq(pzm)*10;
0118 %             nf = 10000;
0119 %             f = linspace(f1, f2, nf);
0120 %             scale = 'lin';
0121 %         end
0122
0123     else
0124         f = varargin{2};
0125     end
0126 elseif nargin == 4
0127     f1 = varargin{2};
0128     f2 = varargin{3};
0129     nf = varargin{4};
0130     f = linspace(f1, f2, nf);
0131 else
0132     error('### incorrect number of inputs.');
```

```

0133 end
0134
0135
0136 % make output plist
0137 f1 = f(1);
0138 f2 = f(end);
0139 nf = length(f);
0140 plo = plist();
0141 plo = append(plo, param('f1', f1));
0142 plo = append(plo, param('f2', f2));
```

```

0143 plo = append(plo, param('nf', nf));
0144 if ~isempty(scale)
0145     plo = append(plo, param('scale', scale));
0146 end
0147
0148 % Now compute the response
0149 r = gain;
0150 for j=1:np
0151     [f, pr] = resp(poles(j),f);
0152 %     figure, loglog(f, abs(pr))
0153     r = r .* pr;
0154 end
0155 for j=1:nz
0156     [f, zr] = resp(zeros(j),f);
0157 %     figure, loglog(f, abs(zr))
0158     r = r .* zr;
0159 end
0160
0161 %----- Build output ao
0162
0163 % create new output fsdata
0164 fs = fsdata(f, r, 1);
0165 fs = set(fs, 'name', sprintf('resp(%s)', pzm.name));
0166
0167 % create new output history
0168 h = history(ALGONAME, VERSION, plo);
0169
0170 % make output analysis object
0171 b = ao(fs, h);
0172
0173 % set name
0174 b = setnh(b, 'name', sprintf('resp(%s)', pzm.name));
0175
0176 % Outputs
0177 if nargout == 0
0178     iplot(b)
0179 elseif nargout == 1
0180     varargout{1} = b;
0181 else
0182     error('### incorrect output arguments');
0183 end
0184
0185
0186 % END

```


[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_Nuttall4b.m

win_Nuttall4b

PURPOSE [^](#)

WIN_NUTTALL4B return Nuttall4b window sample for input z.

SYNOPSIS [^](#)

function w = win_Nuttall4b(z)

DESCRIPTION [^](#)

WIN_NUTTALL4B return Nuttall4b window sample for input z.

M Hewitson 18-05-07

\$Id: win_Nuttall4b.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^](#)

This function calls:

This function is called by:

SOURCE CODE [^](#)

```
0001 function w = win_Nuttall4b(z)
0002
0003 % WIN_NUTTALL4B return Nuttall4b window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Nuttall4b.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.355768 - 0.487396 * cos (z) + 0.144232 * cos (2 * z) - 0.012604 * cos (3
* z);
0012
0013 % Make window struct
0014 w.name      = 'Nuttall4b';
0015 w.alpha    = 0;
0016 w.psll     = 93.3;
0017 w.rov      = 66.3;
```

```
0018 w.nenbw      = 2.0212;  
0019 w.w3db       = 1.9122;  
0020 w.flatness   = -0.8118;  
0021 w.ws         = sum(v);  
0022 w.ws2        = sum(v.*v);  
0023 w.win        = v;  
0024  
0025 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@timespan](#) > char.m

char

PURPOSE

CHAR convert a timespan object into a string.

SYNOPSIS

function pstr = char(ts)

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 1.1 2007/08/17 13:07:15 ingo Exp \$

HISTORY: 23-07-2007 Diepholz
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a timespan object into a string.

This function is called by:

- [char](#) CHAR convert a timespan object into a string.
- [display](#) DISPLAY overloads display functionality for timespan objects.
- [save](#) SAVE a timespan object to file.
- [set](#) SET set timespan properties.
- [subsasgn](#) SUBSASGN define index assignment for timespan properties.
- [timespan](#) TIMESPAN timespan object class constructor.
- [xmladd](#) XMLADD add a timespan object to the given node of the input XML DOM
- [xmlparse](#) XMLPARSE creates a timespan from a timespan xml file.

SOURCE CODE

```

0001 function pstr = char(ts)
0002 % CHAR convert a timespan object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a timespan object into a string.
0007 %
0008 % CALL:          string = char(timespan)
0009 %
0010 % VERSION:       $Id: char.m,v 1.1 2007/08/17 13:07:15 ingo Exp $
0011 %
0012 % HISTORY:       23-07-2007 Diepholz
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 pstr = ['start: ' char(ts.start) ' end: ' char(ts.end)];
0018
0019 % END

```

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

```
0001 function ltpda_allxlabel(label)
0002 % LTPDA_ALLXLABEL Set all the x-axis labels on the current figure.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_ALLXLABEL Set all the x-axis labels on the current figure.
0007 %
0008 % CALL:          ltpda_allxlabel(label)
0009 %
```

```
0010 % VERSION:      $Id: ltpda_allxlabel.m,v 1.5 2007/07/30 16:04:09 ingo Exp $
0011 %
0012 % HISTORY:       17-11-2002 M Hewitson
0013 %                Creation.
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 c = get(gcf, 'children');
0018
0019 for k=1:length(c)
0020
0021     t = get(c(k), 'Tag');
0022     if isempty(t)
0023         h = get(c(k), 'xlabel');
0024         set(h, 'string', label);
0025     end
0026
0027 end
0028
0029 %
0030 % END
```

[Home](#) > [classes](#) > [@zero](#) > resp.m

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(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.

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'

VERSION: \$Id: resp.m,v 1.5 2008/01/11 14:04:44 ingo Exp \$

HISTORY: 03-04-2007 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:

SOURCE CODE

```

0001 function [f,r] = resp(varargin)
0002 % RESP returns the complex response of the zero object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: RESP returns the complex response of the zero object.
0007 %
0008 % CALL:          [f,r] = resp(p, f);           % compute response for vector f
0009 %               [f,r] = resp(p, f1, f2, nf); % compute response from f1 to f2
0010 %                                   % in nf steps.
0011 %               [f,r] = resp(p, pl);          % compute response from parameter
0012 % list.
0013 %
0014 % REMARK:         This is just a helper function. This function should only be
0015 %                 called from class functions.
0016 %
0017 % PARAMETERS:    'f' - a vector of frequencies to evaluate at
0018 %                 or
0019 %                 'f1' - start frequency
0020 %                 'f2' - stop frequency
0021 %                 'nf' - number of evaluation frequencies
0022 %                 'scale' - spacing of frequencies: 'lin' or 'log'
0023 %
0024 % VERSION:       $Id: resp.m,v 1.5 2008/01/11 14:04:44 ingo Exp $
0025 %
0026 % HISTORY:       03-04-2007 Hewitson
0027 %                 Creation
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029 %
0030 VERSION = '$Id: resp.m,v 1.5 2008/01/11 14:04:44 ingo Exp $';
0031
0032 % Check if this is a call for parameters or for the cvs-version number
0033 if nargin == 2
0034     if isa(varargin{1}, 'zero') && ischar(varargin{2})
0035         in = char(varargin{2});
0036         if strcmp(in, 'Params')
0037             f = plist();
0038             return
0039         elseif strcmp(in, 'Version')
0040             f = VERSION;
0041             return
0042         end
0043     end
0044 end

```



```

0045
0046 if nargin < 1
0047     error('### incorrect number of inputs.')
0048 end
0049
0050 % look at inputs
0051 p = varargin{1};
0052 if ~isa(p, 'zero')
0053     error('### first argument should be a zero object.');
```

```

0054 end
0055 % Now look at the zero
0056 f0 = get(p, 'f');
0057 Q  = get(p, 'q');
0058
0059 % check inputs
0060 if nargin == 1
0061
0062     f = linspace(f0/10, 10*f0, 1000);
0063
0064 elseif nargin == 2
0065     if isa(varargin{2}, 'plist')
0066         pl = varargin{2};
0067         f  = find(pl, 'f');
0068         f1 = find(pl, 'f1');
0069         f2 = find(pl, 'f2');
0070         nf = find(pl, 'nf');
0071         scale = find(pl, 'scale');
```

```

0072
0073         if ~isempty(f)
0074             % nothing
0075         elseif ~isempty(f1) && ~isempty(f2) && ~isempty(nf)
0076             if isempty(scale) || strcmp(scale, 'lin')
0077                 f = linspace(f1, f2, nf);
0078             elseif strcmp(scale, 'log')
0079                 f = logspace(log10(f1), log10(f2), nf);
0080             else
0081                 error('### unknown scale option for zero response.');
```

```

0082             end
0083         else
0084             error('### unknown parameter list for zero response.');
```

```

0085         end
0086
0087     else
0088         f = varargin{2};
0089     end
0090 elseif nargin == 4
0091     f1 = varargin{2};
0092     f2 = varargin{3};
0093     nf = varargin{4};
0094     f  = linspace(f1, f2, nf);
0095 else
0096     error('### incorrect number of inputs.');
```

```
0097 end
0098
0099
0100 if Q>0.5
0101     re = 1 - (f.^2./f0^2);
0102     im = f ./ (f0*Q);
0103     r = complex(re, im);
0104 else
0105     re = 1;
0106     im = f./f0;
0107     r = complex(re, im);
0108 end
0109
0110
0111
0112 % END
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_Nuttall4c.m

win_Nuttall4c

PURPOSE

WIN_NUTTALL4C return Nuttall4c window sample for input z.

SYNOPSIS

function w = win_Nuttall4c(z)

DESCRIPTION

WIN_NUTTALL4C return Nuttall4c window sample for input z.

M Hewitson 18-05-07

\$Id: win_Nuttall4c.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_Nuttall4c(z)
0002
0003 % WIN_NUTTALL4C return Nuttall4c window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Nuttall4c.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.3635819 - 0.4891775 * cos (z) + 0.1365995 * cos (2 * z) - 0.0106411 * cos
(3 * z);
0012
0013 % Make window struct
0014 w.name      = 'Nuttall4c';
0015 w.alpha     = 0;
0016 w.psll      = 98.1;
0017 w.rov       = 65.6;
0018 w.nenbw     = 1.9761;
```

```
0019 w.w3db      = 1.8687;  
0020 w.flatness  = -0.8506;  
0021 w.ws        = sum(v);  
0022 w.ws2       = sum(v.*v);  
0023 w.win       = v;  
0024  
0025 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@tsdata](#) > char.m

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 1.3 2007/11/14 18:18:36 ingo 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.
- [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.
- [ne](#) NE overloads the ~= operator for tsdata objects.
- [save](#) SAVE a tsdata object to file.
- [set](#) SET sets a tsdata property.

- [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.
- [xmlparse](#) XMLPARSE creates a tsdata from a tsdata xml file.

SOURCE CODE

```

0001 function pstr = char(p, varargin)
0002 % CHAR convert a param object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a param object into a string.
0007 %
0008 % CALL:          string = char(tsdata)
0009 %
0010 % VERSION:       $Id: char.m,v 1.3 2007/11/14 18:18:36 ingo Exp $
0011 %
0012 % HISTORY:       04-02-2007 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION= '$Id: char.m,v 1.3 2007/11/14 18:18:36 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(p, 'tsdata') && ischar(varargin{1})
0022         in = char(varargin{1});
0023         if strcmp(in, 'Params')
0024             pstr = plist();
0025             return
0026         elseif strcmp(in, 'Version')
0027             pstr = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 pstr = get(p, 'name');
0034
0035 % END

```

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

```
0001 function ltpda_allxscale(scale)
0002 % LTPDA_ALLXSCALE Set all the x scales on the current figure.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_ALLXSCALE Set all the x scales on the current figure.
0007 %
0008 % CALL:           allxscale(scale)       scale = 'lin' or 'log';
0009 %
```

```

0010 % VERSION:      $Id: ltpda_allxscale.m,v 1.5 2007/07/30 16:04:09 ingo Exp $
0011 %
0012 % HISTORY:       02-05-2004 M Hewitson
0013 %                Creation.
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 c = get(gcf, 'children');
0018
0019 if ~(strcmpi(scale, 'lin') || (strcmpi(scale, 'log')))
0020     error('### please use ''lin'' or ''log'' as an input.')
0021 end
0022
0023 for k=1:length(c)
0024
0025     t = get(c(k), 'Tag');
0026     if isempty(t)
0027         set(c(k), 'XScale', scale);
0028     end
0029
0030 end
0031
0032 %
0033 % END

```


[Home](#) > [classes](#) > [@ao](#) > [private](#) > retrieve.m

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:

- [fromxml](#) FROMXML read from part of an XML tree and create an analysis object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
0002
```

```

0003 % RETRIEVE analysis objects with given ids from the LTPDA repository specified
by the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE analysis objects with given ids from the LTPDA
0009 %               repository specified by the input database connection.
0010 %
0011 % CALL:         p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:      $Id: retrieve.m,v 1.9 2007/08/31 17:40:08 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'ao';
0023
0024 %%%%%%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(cccccc.getNodeName), type)
0043                             obj = fromxml(cccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end
0053
0054

```

```
0055  
0056  
0057  
0058  
0059  
0060  
0061 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_Rectangular.m

win_Rectangular

PURPOSE [^](#)

WIN_RECTANGULAR return rectangular window sample for input z.

SYNOPSIS [^](#)

function w = win_Rectangular(z)

DESCRIPTION [^](#)

WIN_RECTANGULAR return rectangular window sample for input z.

M Hewitson 18-05-07

\$Id: win_Rectangular.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^](#)

This function calls:

This function is called by:

SOURCE CODE [^](#)

```
0001 function w = win_Rectangular(z)
0002
0003 % WIN_RECTANGULAR return rectangular window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Rectangular.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 v = ones(size(z));
0011
0012 % Make window struct
0013 w.name      = 'Rectangular';
0014 w.alpha     = 0;
0015 w.psll      = 13.3;
0016 w.rov       = 0.0;
0017 w.nenbw     = 1.0;
0018 w.w3db      = 0.8845;
0019 w.flatness  = -3.9224;
0020 w.ws        = sum(v);
0021 w.ws2       = sum(v.*v);
0022 w.win       = v;
0023
0024 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@xydata](#) > char.m

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 1.3 2007/11/14 18:18:37 ingo 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.
- [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.
- [ne](#) NE overloads the ~= operator for xydata objects.
- [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.
- [xmlparse](#) XMLPARSE creates a xydata from a xydata xml file.

SOURCE CODE

```

0001 function pstr = char(p, varargin)
0002 % CHAR convert a param object into a string.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CHAR convert a param object into a string.
0007 %
0008 % CALL:          string = char(xydata)
0009 %
0010 % VERSION:       $Id: char.m,v 1.3 2007/11/14 18:18:37 ingo Exp $
0011 %
0012 % HISTORY:       04-02-2007 Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION= '$Id: char.m,v 1.3 2007/11/14 18:18:37 ingo Exp $';
0018
0019 % Check if this is a call for parameters
0020 if nargin == 2
0021     if isa(p, 'xydata') && ischar(varargin{1})
0022         in = char(varargin{1});
0023         if strcmp(in, 'Params')
0024             pstr = plist();
0025             return
0026         elseif strcmp(in, 'Version')
0027             pstr = VERSION;
0028             return
0029         end
0030     end
0031 end
0032
0033 pstr = get(p, 'name');
0034
0035
0036 % END

```

[Home](#) > [m](#) > [plottools](#) > ltpda_allyaxis.m

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

```
0001 function ltpda_allyaxis(y1, y2)
0002 % LTPDA_ALLYAXIS Set all the yaxis ranges on the current figure.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_ALLYAXIS Set all the yaxis ranges on the current figure.
0007 %
0008 % CALL: allyaxis(y1, y2)
0009 %
```



```
0010 % VERSION:      $Id: ltpda_allyaxis.m,v 1.5 2007/07/30 16:04:09 ingo Exp $
0011 %
0012 % HISTORY:       17-11-2002 M Hewitson
0013 %                Creation.
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 h = findobj(gcf, 'Type', 'axes', '-not', 'Tag', 'legend');
0018
0019 set(h, 'YLim', [y1 y2])
0020
0021 %
0022 % END
```

[Home](#) > [classes](#) > [@cdata](#) > [private](#) > [retrieve.m](#)

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:

- [fromxml](#) FROMXML read from part of an XML tree and create a cdata object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
```

```

0002
0003 % RETRIEVE cdata objects with given ids from the LTPDA repository specified by
the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE cdata objects with given ids from the LTPDA
0009 % repository specified by the input database connection.
0010 %
0011 % CALL:          p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:       $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 % Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'cdata';
0023
0024 %%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(ccccccc.getNodeName), type)
0043                             obj = fromxml(ccccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end

```

```
0053
0054
0055
0056
0057
0058
0059
0060
0061 % END
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_SFT3F.m

win_SFT3F

PURPOSE

WIN_SFT3F return SFT3F window sample for input z.

SYNOPSIS

function w = win_SFT3F(z)

DESCRIPTION

WIN_SFT3F return SFT3F window sample for input z.

M Hewitson 18-05-07

\$Id: win_SFT3F.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_SFT3F(z)
0002
0003 % WIN_SFT3F return SFT3F window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_SFT3F.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 z = z * 2 * pi;
0011 v = 0.26526 - 0.5 * cos (z) + 0.23474 * cos (2 * z);
0012
0013 % Make window struct
0014 w.name      = 'SFT3F';
0015 w.alpha     = 0;
0016 w.psll      = 31.7;
0017 w.rov       = 66.7;
0018 w.nenbw     = 3.1681;
0019 w.w3db      = 3.1502;
0020 w.flatness  = 0.0082;
0021 w.ws        = sum(v);
0022 w.ws2       = sum(v.*v);
0023 w.win        = v;
0024
0025 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

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

```
0001 function ltpda_allylabel(label)
0002 % LTPDA_ALLYLABEL Set all the y-axis labels on the current figure.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_ALLYLABEL Set all the y-axis labels on the current figure.
0007 %
0008 % CALL:      allylabel(label)
0009 %
```

```
0010 % VERSION:      $Id: ltpda_allylabel.m,v 1.5 2007/07/30 16:04:09 ingo Exp $
0011 %
0012 % HISTORY:       17-11-2002 M Hewitson
0013 %                Creation.
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 c = get(gcf, 'children');
0018
0019 for k=1:length(c)
0020
0021     t = get(c(k), 'Tag');
0022     if isempty(t)
0023         h = get(c(k), 'ylabel');
0024         set(h, 'string', label);
0025     end
0026
0027 end
0028
0029 %
0030 % END
```


[Home](#) > [classes](#) > [@fsdata](#) > [private](#) > [retrieve.m](#)

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:

- [fromxml](#) FROMXML read from part of an XML tree and create a fsdata object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
0002
```

```

0003 % RETRIEVE fsdata objects with given ids from the LTPDA repository specified by
the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE fsdata objects with given ids from the LTPDA
0009 %               repository specified by the input database connection.
0010 %
0011 % CALL:         p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:      $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'fsdata';
0023
0024 %%%%%%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(cccccc.getNodeName), type)
0043                             obj = fromxml(cccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end
0053
0054

```

```
0055  
0056  
0057  
0058  
0059  
0060  
0061 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_SFT3M.m

win_SFT3M

PURPOSE

WIN_SFT3M return SFT3M window sample for input z.

SYNOPSIS

function w = win_SFT3M(z)

DESCRIPTION

WIN_SFT3M return SFT3M window sample for input z.

M Hewitson 18-05-07

\$Id: win_SFT3M.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_SFT3M(z)
0002
0003 % WIN_SFT3M return SFT3M window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_SFT3M.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 z = z * 2 * pi;
0011 v = 0.28235 - 0.52105 * cos (z) + 0.19659 * cos (2 * z);
0012
0013 % Make window struct
0014 w.name      = 'SFT3M';
0015 w.alpha     = 0;
0016 w.psll      = 44.2;
0017 w.rov       = 65.5;
0018 w.nenbw     = 2.9452;
0019 w.w3db      = 2.9183;
0020 w.flatness  = -0.0115;
0021 w.ws        = sum(v);
0022 w.ws2       = sum(v.*v);
0023 w.win        = v;
0024
0025 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

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 1.1 2007/11/28 11:38:46 ingo Exp $

HISTORY:      08-05-07 Diepholz
                  Creation

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

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```

0001 function [aos, pls, invars] = collect_inputs(varargin, in_names)
0002 % COLLECT_INPUTS Collect the AOs and Plists.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: COLLECT_INPUTS Collect the AOs and Plists.
0007 %
0008 % CALL:          [aos, pls, invars] = collect_inputs(varargin, in_names);
0009 %
0010 % INPUTS:        varargin: A list of analysis objects (ao's) and
0011 %                  parameter lists (plist's)
0012 %                  in_names: Corresponding variable names of the contents in
0013 %                  varargin
0014 %
0015 % OUTPUTS:       aos:      Collection of all analysis objects
0016 %                  pls:      Collection of parameter lists
0017 %                  invars: Collection of the ao names of the corresponding ao.
0018 %
0019 % VERSION:       $Id: collect_inputs.m,v 1.1 2007/11/28 11:38:46 ingo Exp $
0020 %
0021 % HISTORY:       08-05-07 Diepholz
0022 %                  Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026
0027 % dd1 = dbstack
0028 % dd2 = dbstatus
0029
0030 aos      = [];
0031 invars   = {};
0032 pls      = plist();
0033
0034 %% collect input ao's, plist's and ao variable names
0035
0036 for j=1:numel(varargin)
0037
0038     if isa(varargin{j}, 'ao')
0039         aos = [aos varargin{j}];
0040
0041         ao_name = in_names{j};
0042         if isempty(ao_name)
0043             ao_name = 'no_ao_name';
0044         end
0045
0046         % Memorise the variable name of the corresponding analysis object.
0047         % If the ao is an array or vector add the index to the variable name
0048         if numel(varargin{j}) == 1
0049             invars{end+1} = ao_name;
0050         else
0051             for ii=1:numel(varargin{j})
0052                 [I,J] = ind2sub(size(varargin{j}),ii);

```

```
0053         invars{end+1} = sprintf('%s(%d,%d)', ao_name, I, J);
0054     end
0055 end
0056 end
0057 if isa(varargin{j}, 'plist')
0058     pls = [pls varargin{j}];
0059 end
0060 end
```

Generated on Tue 22-Jan-2008 10:39:13 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

```
0001 function ltpda_allyscale(scale)
0002 % LTPDA_ALLYSCALE Set all the Y scales on the current figure.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_ALLYSCALE Set all the Y scales on the current figure.
0007 %
0008 % CALL:           allyscale(scale)           scale = 'lin' or 'log';
0009 %
```

```

0010 % VERSION:      $Id: ltpda_allyscale.m,v 1.5 2007/07/30 16:04:09 ingo Exp $
0011 %
0012 % HISTORY:      02-05-2004 M Hewitson
0013 %                Creation.
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 c = get(gcf, 'children');
0018
0019 if ~(strcmpi(scale, 'lin') || (strcmpi(scale, 'log')))
0020     error('### please use ''lin'' or ''log'' as an input.')
0021 end
0022
0023 for k=1:length(c)
0024
0025     t = get(c(k), 'Tag');
0026     if isempty(t)
0027         set(c(k), 'YScale', scale);
0028     end
0029
0030 end
0031
0032 %
0033 % END

```

[Home](#) > [classes](#) > [@history](#) > [private](#) > retrieve.m

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:

- [fromxml](#) FROMXML read from part of an XML tree and create a history object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
```

```

0002
0003 % RETRIEVE history objects with given ids from the LTPDA repository specified
by the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE history objects with given ids from the LTPDA
0009 %                repository specified by the input database connection.
0010 %
0011 % CALL:          p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:       $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'history';
0023
0024 %%%%%%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c    = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(ccccccc.getNodeName), type)
0043                             obj = fromxml(ccccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end

```

```
0053
0054
0055
0056
0057
0058
0059
0060
0061 % END
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_SFT4F.m

win_SFT4F

PURPOSE

WIN_SFT4F return SFT4F window sample for input z.

SYNOPSIS

function w = win_SFT4F(z)

DESCRIPTION

WIN_SFT4F return SFT4F window sample for input z.

M Hewitson 18-05-07

\$Id: win_SFT4F.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_SFT4F(z)
0002
0003 % WIN_SFT4F return SFT4F window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_SFT4F.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.21706 - 0.42103 * cos (z) + 0.28294 * cos (2 * z) - 0.07897 * cos (3 * z);
0012
0013 % Make window struct
0014 w.name      = 'SFT4F';
0015 w.alpha     = 0;
0016 w.psll      = 44.7;
0017 w.rov       = 75;
0018 w.nenbw     = 3.7970;
0019 w.w3db      = 3.7618;
```

```
0020 w.flatness = 0.0041;  
0021 w.ws       = sum(v);  
0022 w.ws2      = sum(v.*v);  
0023 w.win      = v;  
0024  
0025 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [sigproc](#) > [time_domain](#) > ltpda_chkts.m

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

```

0001 function varargout = ltpda_chkts(varargin)
0002 % LTPDA_CHKTS checks that the time-series in the input A0(s) are evenly sampled
according to the sample rate and an allowed error.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_CHKTS checks that the time-series in the input A0(s) are
0007 %              evenly sampled according to the sample rate and an allowed error.
0008 %
0009 % CALL:        res = ltpda_chkts(a)
0010 %              res = 0 - time-stamps samples don't correspond to evenly
sampled data
0011 %              res = 1 - time-stamps correspond to evenly sampled data
0012 %
0013 % GUI CALL:    The following call returns a parameter list object that contains
0014 %              the default parameter values:
0015 %
0016 %              >> pl = ltpda_xcorr('Params')
0017 %
0018 % VERSION:     $Id: ltpda_chkts.m,v 1.4 2007/11/07 15:57:27 ingo Exp $
0019 %
0020 % HISTORY:     02-05-2007 M Hewitson
0021 %              Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024 %
0025 % Check if this is a call for parameters
0026 if nargin == 1
0027     in = char(varargin{1});
0028     if strcmp(in, 'Params')
0029         varargout{1} = getDefaultPL();
0030         return
0031     end
0032 end
0033
0034
0035 % Collect together all A0s from the input variables
0036 as = [];
0037 for j=1:nargin
0038     a = varargin{j};
0039     for k=1:length(a)
0040         ak = a(k);
0041         if isa(ak, 'ao')
0042             as = [as ak];
0043         end
0044     end
0045 end
0046

```

```

0047 na = length(as);
0048
0049 % check each
0050 res = ones(na,1);
0051 for j=1:na
0052     a = as(j);
0053     d = a.data;
0054     if isa(d, 'tsdata')
0055         fs = d.fs;
0056         t = d.x;
0057         tstep = 1/fs;
0058         % Allowed error is 1% of a sample step
0059         Terr = 0.01*tstep;
0060
0061         tt = 0;
0062         % go through time samples
0063         for k=1:length(t)
0064             tdiff = abs(t(k) - tt);
0065             if tdiff > Terr
0066                 res(j) = 0;
0067                 break
0068             end
0069             tt = tt + tstep;
0070         end
0071     else
0072         error('### I only work with time-series AOs at the moment.');
```

```

0073     end
0074
0075 end
0076
0077 varargout{1} = res;
0078
0079
0080
0081 %-----
0082 % Get default params
0083 function plo = getDefaultPL()
0084
0085 disp('* creating default plist...');
0086 plo = plist();
0087 disp('* done.');
```

```

0088
0089 % END
```

[Home](#) > [classes](#) > [@mfir](#) > [private](#) > [retrieve.m](#)

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:

- [fromxml](#) FROMXML read from part of an XML tree and create an mfir object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
```

```

0002
0003 % RETRIEVE mfir objects with given ids from the LTPDA repository specified by
the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE mfir objects with given ids from the LTPDA
0009 % repository specified by the input database connection.
0010 %
0011 % CALL:          p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:       $Id: retrieve.m,v 1.1 2007/08/31 17:40:07 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 % Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'mfir';
0023
0024 %%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(ccccccc.getNodeName), type)
0043                             obj = fromxml(ccccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end

```

```
0053
0054
0055
0056
0057
0058
0059
0060
0061 % END
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_SFT4M.m

win_SFT4M

PURPOSE

WIN_SFT4M return SFT4M window sample for input z.

SYNOPSIS

function w = win_SFT4M(z)

DESCRIPTION

WIN_SFT4M return SFT4M window sample for input z.

M Hewitson 18-05-07

\$Id: win_SFT4M.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_SFT4M(z)
0002
0003 % WIN_SFT4M return SFT4M window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_SFT4M.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 z = z * 2 * pi;
0011 v = 0.241906 - 0.460841 * cos(z) ...
0012     + 0.255381 * cos (2 * z) - 0.041872 * cos (3 * z);
0013
0014 % Make window struct
0015 w.name      = 'SFT4M';
0016 w.alpha     = 0;
0017 w.psl1      = 66.5;
0018 w.rov       = 72.1;
0019 w.nenbw     = 3.3868;
0020 w.w3db      = 3.3451;
0021 w.flatness  = -0.0067;
0022 w.ws        = sum(v);
0023 w.ws2       = sum(v.*v);
0024 w.win       = v;
0025
0026 % END
```

[Home](#) > [m](#) > [sigproc](#) > [frequency_domain](#) > ltpda_cohere.m

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]
 Noverlap - 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')
```

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

CROSS-REFERENCE INFORMATION

This function calls:

- [ltpda_xspect](#) LTPDA_XSPEC performs cross-spectral analysis of various forms.

This function is called by:

SUBFUNCTIONS

- [function plo = getDefaultPL\(\)](#)

SOURCE CODE

```
0001 function varargout = ltpda_cohere(varargin)
0002 % LTPDA_COHERE makes coherence estimates of the time-series objects
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_COHERE makes coherence estimates of the time-series objects
0007 %               in the input analysis objects. Coherences are computed using
0008 %               MATLAB's mscohere (>> help mscohere).
0009 %
0010 % CALL:        b = ltpda_cohere(a1,a2,a3,...,pl)
0011 %
0012 % INPUTS:      b      - output analysis objects
0013 %              aN     - input analysis objects (at least two)
0014 %              pl     - input parameter list
0015 %
0016 %              The function makes coherence estimates between a1 and all other
0017 %              input AOs. Therefore, if the input argument list contains N
0018 %              analysis objects, the output, b, will contain N-1 coherence
0019 %              estimates.
0020 %
0021 %              If the last input argument is a parameter list (plist) it is
0022 %              used.
0023 %              The following parameters are recognised.
0024 %
0025 % PARAMETERS:  Win     - a specwin window object [default: Kaiser -200dB psll]
```

```

0024 %           Noverlap - segment overlap (number of samples) [default: taken from
window function]
0025 %           Nfft    - number of samples in each fft [default: half length of
input data]
0026 %           Order   - order of detrending.
0027 %                   -1 - no detrending
0028 %                   0 - subtract mean [default]
0029 %                   1 - subtract linear fit
0030 %                   N - subtract fit of polynomial, order N
0031 %
0032 %
0033 % VERSION:      $Id: ltpda_cohere.m,v 1.10 2007/12/18 14:11:26 josep Exp $
0034 %
0035 % HISTORY:      07-02-2007 M Hewitson
0036 %                Creation
0037 %
0038 % The following call returns a parameter list object that contains the
0039 % default parameter values:
0040 %
0041 % >> pl = ltpda_cohere('Params')
0042 %
0043 %%%%%%%%%%%%%%%
0044
0045 ALGONAME = mfilename;
0046 VERSION  = '$Id: ltpda_cohere.m,v 1.10 2007/12/18 14:11:26 josep Exp $';
0047
0048 % Check if this is a call for parameters
0049 if nargin == 1
0050     in = char(varargin{1});
0051     if strcmp(in, 'Params')
0052         varargout{1} = getDefaultPL();
0053         return
0054     end
0055 end
0056
0057 % capture input variable names
0058 invars = {};
0059 as      = [];
0060 ps      = [];
0061 for j=1:nargin
0062     if isa(varargin{j}, 'ao')
0063         as = [as varargin{j}];
0064         % record the name of this ao
0065         invars = [invars cellstr(inputname(j))];
0066     end
0067     if isa(varargin{j}, 'plist')
0068         ps = [ps varargin{j}];
0069     end
0070 end
0071
0072
0073 % check plist
0074 if isempty(ps)

```

```
0075     pl = getDefaultPL();
0076 else
0077     pl = combine(ps, getDefaultPL);
0078 end
0079
0080 varargout{1} = ltpda_xspec(as, pl, 'COHERE', ALGONAME, VERSION, invars);
0081
0082
0083 %-----
0084 % Get default params
0085 function plo = getDefaultPL()
0086
0087 disp('* creating default plist...');
0088 plo = plist();
0089 plo = append(plo, param('Nfft', -0.5));
0090 plo = append(plo, param('Win', specwin('Kaiser', 1, 100)));
0091 plo = append(plo, param('Nolap', -1));
0092 plo = append(plo, param('Order', 0));
0093
0094 disp('* done.');
```

[Home](#) > [classes](#) > [@miir](#) > [private](#) > [retrieve.m](#)

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:

- [fromxml](#) FROMXML read from part of an XML tree and create an miir object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
```

```

0002
0003 % RETRIEVE miir objects with given ids from the LTPDA repository specified by
the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE miir objects with given ids from the LTPDA
0009 % repository specified by the input database connection.
0010 %
0011 % CALL:          p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:       $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 % Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'miir';
0023
0024 %%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c    = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(ccccccc.getNodeName), type)
0043                             obj = fromxml(ccccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end

```

```
0053
0054
0055
0056
0057
0058
0059
0060
0061 % END
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_SFT5F.m

win_SFT5F

PURPOSE

WIN_SFT5F return SFT5F window sample for input z.

SYNOPSIS

function w = win_SFT5F(z)

DESCRIPTION

WIN_SFT5F return SFT5F window sample for input z.

M Hewitson 18-05-07

\$Id: win_SFT5F.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_SFT5F(z)
0002
0003 % WIN_SFT5F return SFT5F window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_SFT5F.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.1881 - 0.36923 * cos (z) + ...
0012      0.28702 * cos (2 * z) - 0.13077 * cos (3 * z) + 0.02488 * cos (4 * z);
0013
0014 % Make window struct
0015 w.name      = 'SFT5F';
```

```
0016 w.alpha      = 0;  
0017 w.psll       = 57.3;  
0018 w.rov        = 78.5;  
0019 w.nenbw      = 4.3412;  
0020 w.w3db       = 4.2910;  
0021 w.flatness   = -0.0025;  
0022 w.ws         = sum(v);  
0023 w.ws2        = sum(v.*v);  
0024 w.win        = v;  
0025  
0026 % END
```


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 1.2 2007/11/12 13:41:41 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 %
```

```

0010 % VERSION:      $Id: compute_interval.m,v 1.2 2007/11/12 13:41:41 ingo Exp $
0011 %
0012 % HISTORY:       03-08-2007 Diepholz
0013 %               Creation
0014 %
0015 %%%%%%%%%%%%%%%
0016
0017
0018 number = abs(ts.start.utc_epoch_milli-ts.end.utc_epoch_milli);
0019
0020 form = java.text.SimpleDateFormat;
0021 form.setTimeZone(java.util.TimeZone.getTimeZone('UTC'));
0022
0023 form.applyLocalizedPattern('DD')
0024 num_d = char(form.format(java.util.Date(number)));
0025 num_d = sprintf('%02d', str2num(num_d)-1);
0026
0027 form.applyLocalizedPattern('HH')
0028 num_h = char(form.format(java.util.Date(number)));
0029
0030 form.applyLocalizedPattern('mm')
0031 num_m = char(form.format(java.util.Date(number)));
0032
0033 form.applyLocalizedPattern('ss')
0034 num_s = char(form.format(java.util.Date(number)));
0035
0036 ts.interval = '';
0037
0038 if ~strcmp(num_d, '00')
0039     ts.interval = [ts.interval num_d ' Days '];
0040 end
0041
0042 if ~strcmp(num_h, '00')
0043     ts.interval = [ts.interval num_h ' Hours '];
0044 end
0045
0046 if ~strcmp(num_m, '00')
0047     ts.interval = [ts.interval num_m ' Minutes '];
0048 end
0049
0050 if ~strcmp(num_s, '00')
0051     ts.interval = [ts.interval num_s ' Seconds '];
0052 end
0053
0054
0055

```

[Home](#) > [m](#) > [sigproc](#) > [frequency_domain](#) > ltpda_compute_f.m

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 - minimum 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_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_lpsd` LTPDA_LPSD implement LPSD algorithm for analysis objects.
- `ltpda_ltf` LTPDA_LTFE implement weighted transfer-function estimation computed on a log frequency axis.

SUBFUNCTIONS

- `function pl = getDefaultPL()`

SOURCE CODE

```

0001 function [f,r,m, L,rr,rrr] = ltpda_compute_f(fs, N, Kdes, Kmin, Jdes, olap,
varargin)
0002 % LTPDA_COMPUTE_F compute the frequency and resolution vectors for use with
mlpsd.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_COMPUTE_F compute the frequency and resolution vectors for
0007 %               use with mlpsd.
0008 %
0009 % CALL:         [f,r,m,L] = ltpda_compute_f(fs, N, Kdes, Kmin, Jdes, olap)
0010 %
0011 % INPUTS:       fs    - sample frequency of data
0012 %               N     - length of data vector
0013 %               Kdes  - desired number of averages
0014 %               Kmin  - minimum number of averages
0015 %               Jdes  - desired number of points in spectrum
0016 %               olap  - overlap value (0-1)
0017 %
0018 % VERSION:      $Id: ltpda_compute_f.m,v 1.4 2007/10/28 10:23:40 hewitson Exp $
0019 %
0020 % HISTORY:      22-01-2007 M Hewitson
0021 %               Creation
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 % >> pl = ltpda_compute_f('Params')
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029 %
0030 % Check if this is a call for parameters
0031 if nargin == 1
0032     in = char(fs);
0033     if strcmp(in, 'Params')
0034         f = getDefaultPL();
0035         return

```

```

0036     end
0037 end
0038
0039 if ~isempty(varargin) && strcmp(varargin{1}, 'DEBUG')
0040     %%%%%%%%%% Do linear frequency for testing
0041
0042     f  = [linspace(0, fs/2, Jdes/2 + 1)].';
0043     L  = Jdes * ones(size(f));
0044     r  = fs./L;
0045     m  = f./r;
0046     r  = [];
0047     rr = [];
0048     rrr = [];
0049
0050
0051 else
0052     %%%%%%%%%% Do log frequency
0053
0054     % Settings
0055     fmin = fs/N;
0056     fmax = fs/2;
0057     j    = 1;
0058     f(j) = fmin;
0059
0060     % Initialise constants
0061     rav  = fs/N*(1+(1-olap)*(Kdes-1));
0062     rmin = fs/N*(1+(1-olap)*(Kmin-1));
0063     g    = log(fmax)-log(fmin);
0064
0065     % compute f(j)
0066     while f < fmax
0067         % compute r'(j)
0068         rr(j) = f(j)*g/(Jdes-1);
0069         % compute r''(j)
0070         if rr(j) >= rav
0071             rrr(j) = rr(j);
0072         else
0073             rrr(j) = sqrt(rav*rr(j));
0074         end
0075
0076         % if this is below the lower possible resolution
0077         if rrr(j) <= rmin
0078             rrr(j) = rmin;
0079         end
0080
0081         % compute L(j)
0082         L(j) = round(fs/rrr(j));
0083
0084         % compute freq res
0085         r(j) = fs/L(j);
0086         m(j) = f(j)/r(j);
0087

```

```
0088     tf = f(j) + r(j);
0089     if tf<fmax
0090         f(j+1) = tf;
0091         j = j + 1;
0092     else
0093         break
0094     end
0095 end
0096 end
0097
0098
0099 %-----
0100 % get default parameter list
0101 function pl = getDefaultPL()
0102
0103 pl = plist();
0104 pl = append(pl, param('fs', []));
0105 pl = append(pl, param('N', []));
0106 pl = append(pl, param('Kdes', []));
0107 pl = append(pl, param('Kmin', []));
0108 pl = append(pl, param('Jdes', []));
0109 pl = append(pl, param('olap', []));
0110
0111
```

[Home](#) > [classes](#) > [@param](#) > [private](#) > [retrieve.m](#)

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:

- [fromxml](#) FROMXML read from part of an XML tree and create a param object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
```

```

0002
0003 % RETRIEVE param objects with given ids from the LTPDA repository specified by
the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE param objects with given ids from the LTPDA
0009 % repository specified by the input database connection.
0010 %
0011 % CALL:          p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:       $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 % Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'param';
0023
0024 %%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c    = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(ccccccc.getNodeName), type)
0043                             obj = fromxml(ccccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end

```



```
0053
0054
0055
0056
0057
0058
0059
0060
0061 % END
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_SFT5M.m

win_SFT5M

PURPOSE

WIN_SFT5M return SFT5M window sample for input z.

SYNOPSIS

```
function w = win_SFT5M(z)
```

DESCRIPTION

WIN_SFT5M return SFT5M window sample for input z.

M Hewitson 18-05-07

\$Id: win_SFT5M.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_SFT5M(z)
0002
0003 % WIN_SFT5M return SFT5M window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_SFT5M.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 z = z * 2 * pi;
0011 v = 0.209671 - 0.407331 * cos(z) + ...
0012      0.281225 * cos (2 * z) - 0.092669 * cos (3 * z) + ...
0013      0.0091036 * cos (4 * z);
0014
0015 % Make window struct
0016 w.name      = 'SFT5M';
0017 w.alpha     = 0;
0018 w.psll      = 89.9;
0019 w.rov       = 76;
0020 w.nenbw     = 3.8852;
0021 w.w3db      = 3.8340;
0022 w.flatness  = 0.0039;
0023 w.ws        = sum(v);
0024 w.ws2       = sum(v.*v);
0025 w.win       = v;
0026
0027 % END
```

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')
```

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

CROSS-REFERENCE INFORMATION

This function calls:

- [ltpda_xspect](#) LTPDA_XSPEC performs cross-spectral analysis of various forms.

This function is called by:

SUBFUNCTIONS

- [function plo = getDefaultPL\(\)](#)

SOURCE CODE

```
0001 function varargout = ltpda_cpsd(varargin)
0002 % LTPDA_CPSD makes cross-spectral density estimates of the time-series objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_CPSD makes cross-spectral density estimates of the
0007 %               time-series objects in the input analysis objects. CPSDs are
0008 %               computed
0009 %               using MATLAB's cpsd (>> help cpsd).
0010 % CALL:         b = ltpda_cpsd(a1,a2,a3,...,pl)
0011 %
0012 % INPUTS:       b      - output analysis objects
0013 %               aN     - input analysis objects (at least two)
0014 %               pl     - input parameter list
0015 %
0016 %               The function makes CPSD estimates between a1 and all other input
0017 %               AOs.
0018 %               Therefore, if the input argument list contains N analysis
0019 %               objects, the
0020 %               output a will contain N-1 CPSD estimates.
0021 %               If the last input argument is a parameter list (plist) it is
0022 %               used. The
0023 %               following parameters are recognised.
0024 %               PARAMETERS: 'Win'    - a specwin window object [default: Kaiser -200dB psll]
0025 %               'Nolap' - segment overlap (number of samples) [default: taken
0026 %               from window function]
0027 %               'Nfft'  - number of samples in each fft [default: fs of input
0028 %               data]
0029 %               'Debug' - debug level for terminal output (0-1)
```

```

0027 %
0028 % VERSION:      $Id: ltpda_cpsd.m,v 1.7 2007/07/16 12:52:21 ingo Exp $
0029 %
0030 % HISTORY:      07-02-2007 M Hewitson
0031 %                Creation
0032 %
0033 % The following call returns a parameter list object that contains the
0034 % default parameter values:
0035 %
0036 % >> pl = ltpda_cpsd('Params')
0037 %
0038 %%%%%%%%%%%
0039
0040 ALGONAME = mfilename;
0041 VERSION  = '$Id: ltpda_cpsd.m,v 1.7 2007/07/16 12:52:21 ingo Exp $';
0042
0043
0044 % Check if this is a call for parameters
0045 if nargin == 1
0046     in = char(varargin{1});
0047     if strcmp(in, 'Params')
0048         varargout{1} = getDefaultPL();
0049         return
0050     end
0051 end
0052
0053 % capture input variable names
0054 invars = {};
0055 as      = [];
0056 ps      = [];
0057 for j=1:nargin
0058     invars = [invars cellstr(inputname(j))];
0059     if isa(varargin{j}, 'ao')
0060         as = [as varargin{j}];
0061     end
0062     if isa(varargin{j}, 'plist')
0063         ps = [ps varargin{j}];
0064     end
0065 end
0066
0067
0068 % check plist
0069 if isempty(ps)
0070     pl = getDefaultPL();
0071 else
0072     pl = combine(ps, getDefaultPL);
0073 end
0074
0075 varargout{1} = ltpda_xspec(as, pl, 'CPSD', ALGONAME, VERSION, invars);
0076
0077
0078 %-----

```

```
0079 % Get default params
0080 function plo = getDefaultPL()
0081
0082 disp('* creating default plist...');
0083 plo = plist();
0084 plo = append(plo, param('Nfft', -0.5));
0085 plo = append(plo, param('Win', specwin('Kaiser', 1, 100)));
0086 plo = append(plo, param('Nolap', -1));
0087 disp('* done.');
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@plist](#) > [private](#) > [retrieve.m](#)

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:

- [fromxml](#) FROMXML read from part of an XML tree and create a plist object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
```



```

0002
0003 % RETRIEVE plist objects with given ids from the LTPDA repository specified by
the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE plist objects with given ids from the LTPDA
0009 % repository specified by the input database connection.
0010 %
0011 % CALL:          p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:       $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 % Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'plist';
0023
0024 %%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(ccccccc.getNodeName), type)
0043                             obj = fromxml(ccccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end

```

```
0053
0054
0055
0056
0057
0058
0059
0060
0061 % END
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > win_Welch.m

win_Welch

PURPOSE

WIN_WELCH return Welch window sample for input z.

SYNOPSIS

function w = win_Welch(z)

DESCRIPTION

WIN_WELCH return Welch window sample for input z.

M Hewitson 18-05-07

\$Id: win_Welch.m,v 1.1 2007/05/19 09:04:59 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function w = win_Welch(z)
0002
0003 % WIN_WELCH return Welch window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Welch.m,v 1.1 2007/05/19 09:04:59 hewitson Exp $
0008 %
0009
```

```
0010 v = 1 - (2 * z - 1).^2;  
0011  
0012 % Make window struct  
0013 w.name      = 'Welch';  
0014 w.alpha     = 0;  
0015 w.psll      = 21.3;  
0016 w.rov       = 29.3;  
0017 w.nenbw     = 1.2;  
0018 w.w3db      = 1.1535;  
0019 w.flatness  = -2.2248;  
0020 w.ws        = sum(v);  
0021 w.ws2       = sum(v.*v);  
0022 w.win       = v;  
0023  
0024 % END
```

[Home](#) > [m](#) > [noisegenerator](#) > [private](#) > conv_noisegen.m

conv_noisegen

PURPOSE [^](#)

SYNOPSIS [^](#)

```
function [b,a] = conv_noisegen(pol,zer)
```

DESCRIPTION [^](#)

CROSS-REFERENCE INFORMATION [^](#)

This function calls:

This function is called by:

SOURCE CODE [^](#)

```
0001 function [b,a] = conv_noisegen(pol,zer)
0002
0003 [m,k] = size(pol);
0004 [n,l] = size(zer);
0005
0006 coefb = pol(1,:);
0007
0008 for i = 2:m
0009     coefb = conv(coefb, pol(i,:));
0010 end
0011
0012 b = nonzeros(coefb);
0013
0014 if n~=0
0015     coefa = zer(1,:);
0016     for i = 2:n
0017         coefa = conv(coefa, zer(i,:));
```

```
0018     end
0019     a = nonzeros(coefa);
0020 else
0021     a = 1;
0022 end
0023
0024 %normalize to bn = 1
0025 m = length(b);
0026 normfac = b(m);
0027 b = b/normfac;
0028 a = a/(normfac*sqrt(2));
0029
0030
```

[Home](#) > [m](#) > [plottools](#) > ltpda_cscale.m

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

```
0001 function ltpda_cscale(y1,y2)
0002 % LTPDA_CSCALE Set the color range of the current figure
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_CSCALE Set the color range of the current figure
0007 %
0008 % CALL: cscale(x1,x2)
0009 %
```

```
0010 % VERSION:      $Id: ltpda_cscale.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
0011 %
0012 % HISTORY:       09-10-2001 M Hewitson
0013 %                Creation.
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 set(gca, 'CLim', [y1 y2]);
0018
0019 h = get(gcf, 'Children');
0020
0021 for j=1:length(h)
0022     if strcmp(get(h(j), 'Tag'), 'Colorbar')==1
0023         colorbar;
0024     end
0025 end
0026
0027 % END
```


[Home](#) > [classes](#) > [@pole](#) > [private](#) > [retrieve.m](#)

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:

- [fromxml](#) FROMXML read from part of an XML tree and create a zero object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
```

```

0002
0003 % RETRIEVE pole objects with given ids from the LTPDA repository specified by
the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE pole objects with given ids from the LTPDA
0009 % repository specified by the input database connection.
0010 %
0011 % CALL:          p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:       $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 % Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'pole';
0023
0024 %%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(ccccccc.getNodeName), type)
0043                             obj = fromxml(ccccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end

```

```
0053
0054
0055
0056
0057
0058
0059
0060
0061 % END
```

[Home](#) > [m](#) > [helper](#) > wrapstring.m

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

```

0001 function s = wrapstring(s, n)
0002 % WRAPSTRING wraps a string to a cell array of strings with each cell less than
0003 % n characters long.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: WRAPSTRING wraps a string to a cell array of strings with each
0007 %               cell less than n characters long.
0008 %
0009 % CALL:         s = wrapstring(s, n)
0010 %
0011 % INPUTS:       s - String
0012 %               n - max length of each cell
0013 %
0014 % OUTPUTS:      s - the wrapped cell string
0015 %
0016 % VERSION: $Id: wrapstring.m,v 1.4 2007/07/13 12:17:39 ingo Exp $
0017 %
0018 % HISTORY: 05-02-2007 M Hewitson
0019 %           Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022 %
0023 % convert to a cell if necessary
0024 if ischar(s)
0025     s = [cellstr(s)];
0026 end
0027
0028 % check it makes sense to proceed.
0029 x = splitstring(s{end}, n);
0030 if strcmp(char(x), char(s))
0031     return
0032 end
0033
0034 % now keep splitting until we are happy.
0035 while length(s{end}) >= n
0036     if length(s) > 1
0037         s1 = s(1:end-1);
0038     else
0039         s1 = [];
0040     end
0041     s2 = splitstring(s{end}, n);
0042     if isempty(s2)
0043         break
0044     end
0045     s = [s1 s2];
0046 end
0047
0048 %-----
0049 % split string at first ' ' or ','

```

```
0050 function s = splitstring(s,n)
0051
0052
0053 % disp(sprintf('- splitting %s', s))
0054 fi = 0;
0055 idx = find(s==' ' | s == ',' | s == '(' | s == '=');
0056 if max(idx) > n
0057     for i=1:length(idx)
0058         if idx(i) > n & fi==0
0059             fi = i;
0060         end
0061     end
0062     j = idx(fi);
0063     s = [cellstr(strtrim(s(1:j))) cellstr(strtrim(s(j+1:end)))];
0064 else
0065     s = [];
0066     return;
0067 end
```

[Home](#) > [m](#) > [mysql](#) > ltpda_dbquery.m

ltpda_dbquery

PURPOSE

LTPDA_DBQUERY query an AO repository database.

SYNOPSIS

function varargout = ltpda_dbquery(varargin)

DESCRIPTION

LTPDA_DBQUERY query an AO repository database.

%%%

DESCRIPTION: LTPDA_DBQUERY query an AO repository database.

CALL: info = ltpda_dbquery(server, dbname, table, query);

EXAPLES:

```
>> info = ltpda_dbquery(server, dbname, 'aos', 'id>1000 and id<2000');
>> info = ltpda_dbquery(server, dbname, 'aos', 'name like "x12"');
>> info = ltpda_dbquery(server, dbname, 'AOUsers', 'name="aouser"');
>> info = ltpda_dbquery(server, dbname, 'collections', 'UserID=3');
>> info = ltpda_dbquery(server, dbname, 'collections', 'aoids="1 2"');
>> info = ltpda_dbquery(server, dbname, 'transactions', 'UserID=3');
>> info = ltpda_dbquery(server, dbname, 'transactions', 'aoid=56');
```

The returned 'info' structure contains the fields from each mathcing record.

```
>> info = ltpda_dbquery(server, dbname)
```

The 'info' structure will contain a list of the tables in the database.

```
>> info = ltpda_dbquery(server, dbname, query)
```

The 'info' structure will contain a list of records resulting from the SQL query.

VERSION: \$Id: ltpda_dbquery.m,v 1.4 2007/11/13 13:53:47 hewitson Exp \$

HISTORY: 10-05-2007 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [mysql_connect](#) MYSQL_CONNECT connects to an AO repository and returns the connection object.

This function is called by:

SUBFUNCTIONS

- function info = getTableList(hostname, dbname)
- function info = getFieldList(table)
- function info = simpleQuery(hostname, dbname, q)
- function info = runQuery(hostname, dbname, table, query)

SOURCE CODE

```

0001 function varargout = ltpda_dbquery(varargin)
0002 % LTPDA_DBQUERY query an AO repository database.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_DBQUERY query an AO repository database.
0007 %
0008 % CALL:          info = ltpda_dbquery(server, dbname, table, query);
0009 %
0010 % EXAPLES:
0011 %
0012 %   >> info  = ltpda_dbquery(server, dbname, 'aos',          'id>1000 and
id<2000');
0013 %   >> info  = ltpda_dbquery(server, dbname, 'aos',          'name like "x12"');
0014 %   >> info  = ltpda_dbquery(server, dbname, 'AOusers',      'name="aouser"');
0015 %   >> info  = ltpda_dbquery(server, dbname, 'collections',  'UserID=3');
0016 %   >> info  = ltpda_dbquery(server, dbname, 'collections',  'aoids="1 2"');
0017 %   >> info  = ltpda_dbquery(server, dbname, 'transactions', 'UserID=3');
0018 %   >> info  = ltpda_dbquery(server, dbname, 'transactions', 'aoid=56');
0019 %
0020 %   The returned 'info' structure contains the fields from each mathcing
0021 %   record.
0022 %
0023 %   >> info  = ltpda_dbquery(server, dbname)
0024 %
0025 %   The 'info' structure will contain a list of the tables in the database.
0026 %
0027 %   >> info  = ltpda_dbquery(server, dbname, query)
0028 %
0029 %   The 'info' structure will contain a list of records resulting from the SQL
query.
0030 %

```



```

0031 % VERSION:      $Id: ltpda_dbquery.m,v 1.4 2007/11/13 13:53:47 hewitson Exp $
0032 %
0033 % HISTORY:      10-05-2007 M Hewitson
0034 %              Creation
0035 %
0036 %%%%%%%%%%%%%%%
0037
0038 % get table list
0039 if nargin == 2
0040
0041     info = getTableList(varargin{1}, varargin{2});
0042
0043 % get field list
0044 elseif nargin == 3
0045
0046 %     info = getFieldList(varargin{1});
0047     info = simpleQuery(varargin{1}, varargin{2}, varargin{3});
0048
0049 elseif nargin == 4
0050
0051     hostname = varargin{1};
0052     dbname   = varargin{2};
0053     table    = varargin{1};
0054     query    = varargin{2};
0055
0056     info = runQuery(hostname, dbname, table, query);
0057
0058 else
0059     error('### Incorrect inputs. ');
0060 end
0061
0062 varargout{1} = info;
0063
0064 %-----
0065 % Get table list
0066 function info = getTableList(hostname, dbname)
0067
0068 % open a connection
0069 conn = mysql_connect(hostname, dbname);
0070 curs = exec(conn, 'show tables');
0071 curs = fetch(curs);
0072 info = curs.Data;
0073 close(curs);
0074
0075 % Close MySQL connection
0076 close(conn);
0077
0078 %-----
0079 % Get field list
0080 function info = getFieldList(table)
0081
0082 % open a connection

```

```

0083 conn = mysql_connect();
0084
0085 curs = exec(conn, sprintf('describe %s', table));
0086 curs = fetch(curs);
0087 info = curs.Data;
0088 close(curs);
0089
0090 % Close MySQL connection
0091 close(conn);
0092
0093 %-----
0094 % Get field list
0095 function info = simpleQuery(hostname, dbname, q)
0096
0097 % open a connection
0098 conn = mysql_connect(hostname, dbname);
0099
0100 curs = exec(conn, sprintf('%s', q));
0101 curs = fetch(curs);
0102 info = curs.Data;
0103 close(curs);
0104
0105 % Close MySQL connection
0106 close(conn);
0107
0108
0109 %-----
0110 % Run a query
0111 function info = runQuery(hostname, dbname, table, query)
0112
0113 % open a connection
0114 conn = mysql_connect(hostname, dbname);
0115
0116 % Run query
0117 info = [];
0118
0119 fieldlist = getFieldList(table);
0120 fields = fieldlist(:,1);
0121 f = '';
0122 fs = {};
0123 for j=1:length(fields)
0124     % special cases
0125     switch table
0126     case 'aos'
0127         if ~strcmp(fields{j}, 'AO')
0128             f=[f fields{j} ', '];
0129             fs = [fs fields{j}];
0130         end
0131     otherwise
0132         f=[f fields{j} ', '];
0133         fs = [fs fields{j}];
0134     end

```

```
0135 end
0136 q = sprintf('select %s from %s where %s', f(1:end-1), table, query);
0137 disp(['** QUERY: ' q]);
0138 curs = exec(conn, q);
0139 curs = fetch(curs);
0140 info = curs.Data;
0141 close(curs);
0142
0143 % Close MySQL connection
0144 close(conn);
0145
0146
0147 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@provenance](#) > [private](#) > retrieve.m

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:

- [fromxml](#) FROMXML read from part of an XML tree and create a provenance object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
```

```

0002
0003 % RETRIEVE provenance objects with given ids from the LTPDA repository
specified by the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE provenance objects with given ids from the LTPDA
0009 % repository specified by the input database connection.
0010 %
0011 % CALL:          p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:       $Id: retrieve.m,v 1.1 2007/08/31 17:40:09 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'provenance';
0023
0024 %%%%%%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c    = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(cccccc.getNodeName), type)
0043                             obj = fromxml(cccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end

```

```
0053
0054
0055
0056
0057
0058
0059
0060
0061 % END
```

[Home](#) > [classes](#) > [@ao](#) > [private](#) > xdoc2ao.m

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

```
0001 function aout = xdoc2ao(xdoc)
0002
0003 % XDOC2AO converts a DOM object to an AO.
0004 %
0005 % M Hewitson 16-05-07
0006 %
0007 % $Id: xdoc2ao.m,v 1.2 2007/11/14 16:30:18 ingo Exp $
0008 %
0009
```

```

0010 ain = xdoc.getElementsByTagName('a');
0011 a = ain.item(0);
0012
0013 children = a.getChildNodes;
0014
0015 name      = '';
0016 prov      = [];
0017 description= '';
0018 mfile     = '';
0019 mfname    = '';
0020 mdlfile   = '';
0021 mdlfname  = '';
0022 hist      = [];
0023 data      = [];
0024
0025 % Go through children of AO object
0026 for j=1:children.getLength
0027
0028     ch = children.item(j-1);
0029
0030     nodeName = char(ch.getNodeName);
0031     if ischar(nodeName)
0032         switch nodeName
0033             case 'Name'
0034                 name = char(ch.getTextContent);
0035             case 'Provenance'
0036                 prov = xml_read_provenance(ch);
0037             case 'Description'
0038                 description = char(ch.getTextContent);
0039             case 'M-file'
0040                 mfile = char(ch.getTextContent);
0041             case 'Mfilename'
0042                 mfname = char(ch.getTextContent);
0043             case 'Mdl-file'
0044                 mdlfile = char(ch.getTextContent);
0045             case 'Mdlfilename'
0046                 mdlfname = char(ch.getTextContent);
0047             case 'hist'
0048                 hist = xml_read_hist(ch);
0049             case 'Data'
0050                 data = xml_read_data(ch);
0051             otherwise
0052                 warning(['!!! unknown child node: ' nodeName]);

```



```
0053     end
0054   end
0055 end
0056
0057 % Now build AO
0058 aout.name      = name;
0059 aout.data      = data;
0060 aout.hist      = hist;
0061 aout.provenance = prov;
0062 aout.description = description;
0063 aout.mfile     = mfile;
0064 aout.mfilename = mfname;
0065 aout.mdlfile   = mdlfile;
0066 aout.mdlfilename = mdlfname;
0067 aout = class(aout, 'ao');
0068
0069
```

[Home](#) > [classes](#) > [@pole](#) > cp2iir.m

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 1.4 2008/01/11 13:42:41 ingo 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

```

0001 function [a,b] = cp2iir(p, fs)
0002 % CP2IIR Return a,b IIR filter coefficients for a complex pole designed using
the bilinear transform.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CP2IIR Return a,b IIR filter coefficients for a complex pole
0007 %               designed using the bilinear transform.
0008 %
0009 % CALL:        [a,b] = cp2iir(p, fs)
0010 %
0011 % REMARK:      This is just a helper function. This function should only be
0012 %               called from class functions.
0013 %
0014 % INPUT:       p - pole object
0015 %               fs - the sample rate for the filter
0016 %
0017 % VERSION:     $Id: cp2iir.m,v 1.4 2008/01/11 13:42:41 ingo Exp $
0018 %
0019 % HISTORY:     03-04-2007 M Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: cp2iir.m,v 1.4 2008/01/11 13:42:41 ingo Exp $';
0025
0026 %%%% 'Params' && 'Version' Call %%%%
0027 if nargin == 2
0028     if isa(p, 'pole') && ischar(fs)
0029         in = fs;
0030         if strcmp(in, 'Params')
0031             a = plist();
0032             return
0033         elseif strcmp(in, 'Version')
0034             a = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 f0 = get(p, 'f');
0041 q  = get(p, 'q');
0042
0043 w0  = f0*2*pi;
0044 w02 = w0^2;
0045
0046 k    = (q*w02 + 4*q*fs*fs + 2*w0*fs) / (q*w02);
0047 b(1) = 1;
0048 b(2) = (2*w02-8*fs*fs) / (k*w02);
0049 b(3) = (q*w02 + 4*q*fs*fs - 2*w0*fs) / (k*q*w02);
0050
0051 a(1) = 1/k;

```

```
0052 a(2) = -2/k;  
0053 a(3) = -1/k;  
0054 a      = a*-2;  
0055  
0056 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [helper](#) > ltpda_deg2rad.m

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 [^](#)

```
0001 function r = ltpda_deg2rad(deg, min, sec)
0002 % LTPDA_DEG2RAD Convert degrees to radians
0003 %
```

```
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_DEG2RAD Convert degrees to radians
0007 %
0008 % CALL:          r = ltpda_deg2rad(deg, min, sec)
0009 %
0010 % INPUTS:        deg - degrees
0011 %                min - minutes
0012 %                sec - seconds
0013 %
0014 % OUTPUTS:       r    - radians
0015 %
0016 % VERSION: $Id: ltpda_deg2rad.m,v 1.5 2007/07/13 12:17:39 ingo Exp $
0017 %
0018 % HISTORY: 26-01-2007 M Hewitson
0019 %          Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 r = (deg + min/60 + sec/3600)*pi/180;
0024
0025
```

[Home](#) > [classes](#) > [@pzmodel](#) > [private](#) > [retrieve.m](#)

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:

- [fromxml](#) FROMXML read from part of an XML tree and create a pzmodel object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
```

```

0002
0003 % RETRIEVE pzmodel objects with given ids from the LTPDA repository specified
by the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE pzmodel objects with given ids from the LTPDA
0009 % repository specified by the input database connection.
0010 %
0011 % CALL:          p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:       $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'pzmodel';
0023
0024 %%%%%%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c    = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(ccccccc.getNodeName), type)
0043                             obj = fromxml(ccccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end

```



```
0053
0054
0055
0056
0057
0058
0059
0060
0061 % END
```

[Home](#) > [mex](#) > ltpda_dft.m

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 20:20:41 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 % LTPDA_DFT computes the DFT of a signal at one frequency.
0002 %
0003 % M Hewitson 15-01-08
0004 %
0005 % $Id: ltpda_dft.m,v 1.1 2008/01/15 20:20:41 hewitson Exp $
0006 %
```

[Home](#) > [classes](#) > [@specwin](#) > [private](#) > retrieve.m

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:

- [fromxml](#) FROMXML read from part of an XML tree and create a specwin object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
```

```

0002
0003 % RETRIEVE specwin objects with given ids from the LTPDA repository specified
by the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE specwin objects with given ids from the LTPDA
0009 %                repository specified by the input database connection.
0010 %
0011 % CALL:         p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:      $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'specwin';
0023
0024 %%%%%%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c    = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(ccccccc.getNodeName), type)
0043                             obj = fromxml(ccccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end

```

```
0053
0054
0055
0056
0057
0058
0059
0060
0061 % END
```

[Home](#) > [classes](#) > [@cdata](#) > xml.m

xml

PURPOSE

XML convert a cdata object to an DOM XML object.

SYNOPSIS

function x = xml(d)

DESCRIPTION

XML convert a cdata object to an DOM XML object.

%%%

DESCRIPTION: XML convert a cdata object to an DOM XML object.

CALL: >> x = xml(d)

VERSION: \$Id: xml.m,v 1.2 2007/09/15 04:50:19 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add a cdata object to the given node of the input XML DOM

This function is called by:

- [xmladd](#) XMLADD add a cdata object to the given node of the input XML DOM

SOURCE CODE

```
0001 function x = xml(d)
0002 % XML convert a cdata object to an DOM XML object.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert a cdata object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(d)
0009 %
0010 % VERSION: $Id: xml.m,v 1.2 2007/09/15 04:50:19 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('cdata');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(d, x, 'Object', x.docRootNode);
0025
0026
0027 % END

```

[Home](#) > [classes](#) > [@zero](#) > cz2iir.m

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 1.4 2008/01/11 13:42:41 ingo 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 [^](#)


```

0001 function [a,b] = cz2iir(z, fs)
0002 % CZ2IIR return a,b IIR filter coefficients for a complex zero designed using
the bilinear transform.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CZ2IIR return a,b IIR filter coefficients for a complex zero
0007 %               designed using the bilinear transform.
0008 %
0009 % CALL:        [a,b] = cz2iir(z, fs)
0010 %
0011 % REMARK:      This is just a helper function. This function should only be
0012 %               called from class functions.
0013 %
0014 % INPUTS:      z - zero object
0015 %               fs - the sample rate for the filter
0016 %
0017 % VERSION:     $Id: cz2iir.m,v 1.4 2008/01/11 13:42:41 ingo Exp $
0018 %
0019 % HISTORY:     03-04-2007 Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: cz2iir.m,v 1.4 2008/01/11 13:42:41 ingo Exp $';
0025
0026 %%%% 'Params' && 'Version' Call %%%%
0027 if nargin == 2
0028     if isa(z, 'zero') && ischar(fs)
0029         in = fs;
0030         if strcmp(in, 'Params')
0031             a = plist();
0032             return
0033         elseif strcmp(in, 'Version')
0034             a = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 f0 = get(z, 'f');
0041 q = get(z, 'q');
0042
0043 w0 = f0*2*pi;
0044 w02 = w0^2;
0045
0046 a(1) = (-q*w02/2 - 2*q*fs*fs - w0*fs) / (q*w02);
0047 a(2) = (-w02+4*fs*fs) / w02;
0048 a(3) = (-q*w02/2 - 2*q*fs*fs + w0*fs) / (q*w02);
0049
0050 b(1) = 1;
0051 b(2) = -2;

```

```
0052 b(3) = -1;  
0053  
0054 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [mex](#) > [macintel](#) > [2007b](#) > ltpda_dft.m

ltpda_dft

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 20:20:41 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 % LTPDA_DFT computes the DFT of a signal at one frequency.
0002 %
0003 % M Hewitson 15-01-08
0004 %
0005 % $Id: ltpda_dft.m,v 1.1 2008/01/15 20:20:41 hewitson Exp $
0006 %
```

[Home](#) > [classes](#) > [@time](#) > [private](#) > [retrieve.m](#)

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:

- [fromxml](#) FROMXML read from part of an XML tree and create a time object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
```

```

0002
0003 % RETRIEVE time objects with given ids from the LTPDA repository specified by
the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE time objects with given ids from the LTPDA
0009 %                repository specified by the input database connection.
0010 %
0011 % CALL:          p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:       $Id: retrieve.m,v 1.1 2007/08/31 17:40:07 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'time';
0023
0024 %%%%%%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c    = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(ccccccc.getNodeName), type)
0043                             obj = fromxml(ccccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end

```

```
0053
0054
0055
0056
0057
0058
0059
0060
0061 % END
```

[Home](#) > [classes](#) > [@fsdata](#) > xml.m

xml

PURPOSE

XML convert an fsdata object to an DOM XML object.

SYNOPSIS

function x = xml(d)

DESCRIPTION

XML convert an fsdata object to an DOM XML object.

%%%

DESCRIPTION: XML convert an fsdata object to an DOM XML object.

CALL: >> x = xml(d)

VERSION: \$Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add a fsdata object to the given node of the input XML DOM

This function is called by:

- [xmladd](#) XMLADD add a fsdata object to the given node of the input XML DOM

SOURCE CODE

```
0001 function x = xml(d)
0002 % XML convert an fsdata object to an DOM XML object.
0003 %
```

```
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert an fsdata object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(d)
0009 %
0010 % VERSION: $Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('fsdata');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(d, x, 'Object', x.docRootNode);
0025
0026
0027 % END
```


[Home](#) > [mex](#) > [windows](#) > [2007b](#) > ltpda_dft.m

ltpda_dft

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 20:20:41 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 % LTPDA_DFT computes the DFT of a signal at one frequency.
0002 %
0003 % M Hewitson 15-01-08
0004 %
0005 % $Id: ltpda_dft.m,v 1.1 2008/01/15 20:20:41 hewitson Exp $
0006 %
```

[Home](#) > [classes](#) > [@timeformat](#) > [private](#) > [retrieve.m](#)

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:

- [fromxml](#) FROMXML read from part of an XML tree and create a timeformat object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
```

```

0002
0003 % RETRIEVE timeformat objects with given ids from the LTPDA repository
specified by the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE timeformat objects with given ids from the LTPDA
0009 % repository specified by the input database connection.
0010 %
0011 % CALL:          p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:       $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'timeformat';
0023
0024 %%%%%%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c    = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(ccccccc.getNodeName), type)
0043                             obj = fromxml(ccccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end

```

```
0053
0054
0055
0056
0057
0058
0059
0060
0061 % END
```

[Home](#) > [classes](#) > [@history](#) > xml.m

xml

PURPOSE

XML convert a specwin object to an DOM XML object.

SYNOPSIS

function x = xml(s)

DESCRIPTION

XML convert a specwin object to an DOM XML object.

%%%

DESCRIPTION: XML convert a specwin object to an DOM XML object.

CALL: >> x = xml(s)

VERSION: \$Id: xml.m,v 1.3 2007/09/15 04:50:18 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add an history object to the given node of the input XML DOM

This function is called by:

- [xmladd](#) XMLADD add an history object to the given node of the input XML DOM

SOURCE CODE

```
0001 function x = xml(s)
0002 % XML convert a specwin object to an DOM XML object.
0003 %
```

```
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert a specwin object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(s)
0009 %
0010 % VERSION: $Id: xml.m,v 1.3 2007/09/15 04:50:18 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('history');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(s, x, 'Object', x.docRootNode);
0025
0026
0027 % END
```

[Home](#) > [m](#) > [analyses](#) > [stuttgart](#) > ltpda_diff_acc.m

ltpda_diff_acc

PURPOSE

LTPDA_DIFF_ACC

SYNOPSIS

function [X1, X2, X3, P] = ltpda_diff_acc(varargin)

DESCRIPTION

LTPDA_DIFF_ACC

Usage: [X1, X2, X3, P] = ltpda_diff_acc(sigX1, sigX12, sigFsus, pl)

Bob 21-05-07

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SUBFUNCTIONS

- function [x1, x2, x3, p] = kalman(fs, sigx1, sigx12, sigfsus, p1)
- function pl = getDefaultPlist()

SOURCE CODE

```
0001 function [X1, X2, X3, P] = ltpda_diff_acc(varargin)
0002
0003 % LTPDA_DIFF_ACC
0004 %
0005 % Usage: [X1, X2, X3, P] = ltpda_diff_acc(sigX1, sigX12, sigFsus, pl)
0006 %
0007 %
0008 % Bob 21-05-07
0009 %
0010
0011 % Check if this is a call for parameters (necessary forthe gui)
0012 if nargin == 1
```

```

0013     if ischar(varargin{1})
0014         if strcmp(varargin{1}, 'Params')
0015             X1 = getDefaultPlist();
0016             return
0017         end
0018     end
0019 end
0020
0021
0022 % capture input variable names
0023 invars = {};
0024 for j=1:nargin
0025     if isa(varargin{j}, 'ao')
0026         invars = [invars cellstr(inputname(j))];
0027     end
0028 end
0029
0030 ALGONAME = mfilename;
0031 VERSION  = '$Id: ltpda_diff_acc.m,v 1.3 2007/11/07 15:57:27 ingo Exp $';
0032
0033 % Get input signals
0034 sigX1    = varargin{1};
0035 sigX12   = varargin{2};
0036 sigFsus  = varargin{3};
0037 pl       = varargin{4};
0038
0039 % Combine plists
0040 pl = combine(pl, getDefaultPlist());
0041
0042 % Get parameters from plist
0043 p1 = find(pl, 'p1');
0044
0045 % Get data from analysis objects
0046 fs    = sigX12.data.fs;
0047 sigx12 = sigX12.data.y;
0048 sigx1  = sigX1.data.y;
0049 sigfsus = sigFsus.data.y;
0050
0051 % Call Kalman thingy
0052 [x1, x2, x3, p] = kalman(fs, sigx1, sigx12, sigfsus, p1);
0053
0054 % History for output AOs
0055 inhist = [sigX1.hist sigX12.hist sigFsus.hist];
0056 h = history(ALGONAME, VERSION, pl, inhist);
0057 h = set(h, 'invars', invars);
0058
0059 % Make output AOs
0060 ts = tsdata(sigX1.data.x, x1);
0061 ts = set(ts, 'name', 'x1');
0062 X1 = ao(ts, h);
0063 X1 = set(X1, 'name', 'X1');
0064

```



```

0065 ts = tsdata(sigX1.data.x, x2);
0066 ts = set(ts, 'name', 'x2');
0067 X2 = ao(ts, h);
0068 X2 = set(X2, 'name', 'X2');
0069
0070 ts = tsdata(sigX1.data.x, x3);
0071 ts = set(ts, 'name', 'x3');
0072 X3 = ao(ts, h);
0073 X3 = set(X3, 'name', 'X3');
0074
0075 ts = cdata(p);
0076 ts = set(ts, 'name', 'P');
0077 P = ao(ts, h);
0078 P = set(P, 'name', 'P');
0079
0080 %-----
0081 % Apply Kalman filter
0082 function [x1, x2, x3, p] = kalman(fs, sigx1, sigx12, sigfsus, p1)
0083
0084 % x1 = randn(length(sigx1),1);
0085 % x2 = randn(length(sigx12),1);
0086 % x3 = randn(length(sigfsus),1);
0087 % p = randn(3,3,length(sigfsus));
0088
0089 t = 1/fs;
0090
0091 %define variables
0092 om2 = sqrt(2.24*(10^(-6)));
0093 deltaom = -7*(10^(-7));
0094
0095 sigma1_quad = 0.675*(10^(-10));
0096 sigma2_quad = 0.417*(10^(-9));
0097 sigma3_quad = 0.5*(10^(-9));
0098 sigmay_quad = 0.675*(10^(-10));
0099 R = sigmay_quad;
0100
0101 %matrices and vectors
0102
0103 PHI = [cosh(om2*t)      (1/om2)*sinh(om2*t)  -(1/(om2^2))*(1-cosh(om2*t));
0104        om2*sinh(om2*t)  cosh(om2*t)          (1/om2)*sinh(om2*t);
0105        0                0                    1
0106        ];
0107 Bd = [(-1/(om2^2))*(cosh(om2*t)-1)*(-deltaom)  (1/(om2^2))*(cosh(om2*t))-1;
0108       (1/om2)*sinh(om2*t)*(-deltaom)          (1/om2)*sinh(om2*t);
0109       0                                         0
0110        ];
0111 H = [1 0 0];
0112
0113 x0 = [0 0 0]';
0114
0115 P0 = [0.0001 0 0;
0116       0 0.0001 0;

```

```

0117     0  0  0.0001];
0118
0119 %Variablen zur Berechnung von Q(k)
0120 alpha1 = (sigma1_quad/2) -(sigma2_quad/(2*(om2^2))) +(3*sigma3_quad/(2*
(om2^4)));
0121 alpha2 = (sigma1_quad/4) +(sigma2_quad/(4*(om2^2))) +(sigma3_quad/(4*(om2^4)));
0122 beta = (1/4)*(sigma1_quad*om2 +sigma2_quad/om2 + sigma3_quad/(om2^3));
0123 epsi1 = sigma2_quad/2 -(sigma1_quad*(om2^2))/2 - sigma3_quad/(2*(om2^2));
0124 epsi2 = (sigma1_quad*(om2^2))/4 + sigma2_quad/4 + sigma3_quad/(4*(om2^2));
0125
0126 Q11 = alpha1*t +(exp(-om2*t)-exp(om2*t))*(sigma3_quad/(om2^5)) +(exp(2*om2*t)-
exp(-2*om2*t))*(alpha2/(2*om2));
0127
0128 Q12 = -beta/om2 +sigma3_quad/(om2^4) +(beta/(2*om2))*(exp(2*om2*t)+exp(-
2*om2*t)) -(sigma3_quad/(2*(om2^4)))*(exp(-om2*t)+exp(om2*t));
0129
0130 Q13 = -(sigma3_quad/(om2^2))*t+(sigma3_quad/(2*(om2^3)))*(-exp(-om2*t)+exp
(om2*t));
0131
0132 Q21 = Q12;
0133
0134 Q22 = epsi1*t+(epsi2/(2*om2))*(exp(2*om2*t)-exp(-2*om2*t));
0135
0136 Q23 = -sigma3_quad/(om2^2) +(sigma3_quad/(2*(om2^2)))*(exp(-om2*t)+exp(om2*t));
0137
0138 Q31 = Q13;
0139
0140 Q32 = Q23;
0141
0142 Q33 = sigma3_quad*t;
0143
0144 Q = [Q11  Q12  Q13;
0145      Q21  Q22  Q23;
0146      Q31  Q32  Q33];
0147
0148 x1 = zeros(size(sigx12));
0149 x2 = zeros(size(sigx12));
0150 x3 = zeros(size(sigx12));
0151 p  = zeros(3,3,length(sigx12));
0152
0153 u = [sigx1 sigfsus];
0154 size(sigx1)
0155 size(sigfsus)
0156 size(u)
0157
0158 for i=1:length(x1)
0159
0160     % kalman gain
0161     K = P0*H'*inv(H*P0*H'+R);
0162     % new estimation
0163     x_est = x0 + K*(sigx12(i)-H*x0);
0164     x1(i) = x_est(1,1);

```

```
0165     x2(i) = x_est(2,1);
0166     x3(i) = x_est(3,1);
0167     % new covariance
0168     P = (eye(3) - K*H)*P0;
0169     p(:, :, i) = P;
0170
0171     % updated state estimate
0172     x0 = PHI*x_est + Bd*u(i, :)';
0173     P0 = PHI*P*PHI' + Q;
0174 %     i = i+1;
0175
0176 end
0177
0178 %-----
0179 % get default parameter list
0180 function pl = getDefaultPlist()
0181
0182 pl = plist();
0183 pl = append(pl, param('p1', 1));
0184
0185
```

[Home](#) > [classes](#) > [@timespan](#) > [private](#) > [retrieve.m](#)

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:

- [fromxml](#) FROMXML read from part of an XML tree and create a timespan object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
0002
```

```

0003 % RETRIEVE timespan objects with given ids from the LTPDA repository specified
by the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE timespan objects with given ids from the LTPDA
0009 %               repository specified by the input database connection.
0010 %
0011 % CALL:         p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:      $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'timespan';
0023
0024 %%%%%%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c    = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(cccccc.getNodeName), type)
0043                             obj = fromxml(cccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end
0053
0054

```

```
0055  
0056  
0057  
0058  
0059  
0060  
0061 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@mfir](#) > xml.m

xml

PURPOSE

XML convert an mfir object to an DOM XML object.

SYNOPSIS

function x = xml(f)

DESCRIPTION

XML convert an mfir object to an DOM XML object.

%%%

DESCRIPTION: XML convert an mfir object to an DOM XML object.

CALL: >> x = xml(f)

VERSION: \$Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add an MFIR object to the given node of the input XML DOM

This function is called by:

- [xmladd](#) XMLADD add an MFIR object to the given node of the input XML DOM

SOURCE CODE

```
0001 function x = xml(f)
0002 % XML convert an mfir object to an DOM XML object.
0003 %
```

```
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert an mfir object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(f)
0009 %
0010 % VERSION: $Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('mfir');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(f, x, 'Object', x.docRootNode);
0025
0026
0027 % END
```


[Home](#) > [m](#) > [analyses](#) > [stuttgart](#) > ltpda_dkalman.m

ltpda_dkalman

PURPOSE

LTPDA_DKALMAN

SYNOPSIS

function [X1, X2, X3] = ltpda_dkalman(varargin)

DESCRIPTION

LTPDA_DKALMAN

Usage: [X1, X2, X3] = ltpda_dkalman(Ad,Bd,Cd,Dd,Q,R,P0,x0,in1,in2,meas)

Ad: discrete system matrix [3x3]

Bd: discrete input matrix [3x2]

Cd: discrete measurement matrix [1x3]

Dd: 0

Q: discrete covariance matrix of process noise [3x3]

R: discrete covariance matrix of measurement noise [1x1]

x0: prior state estimate

P0: error covariance for x0

in1: first input ao

in2: second input ao

meas: measurement data (ao)

veronika 03-07-07

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [test_dkalman](#) %%%%%%%%%%

SUBFUNCTIONS

- `function pl = getDefaultPlist()`

SOURCE CODE

```

0001 function [X1, X2, X3] = ltpda_dkalman(varargin)
0002
0003 % LTPDA_DKALMAN
0004 %
0005 % Usage: [X1, X2, X3] = ltpda_dkalman(Ad,Bd,Cd,Dd,Q,R,P0,x0,in1,in2,meas)
0006 %
0007 %     Ad: discrete system matrix [3x3]
0008 %     Bd: discrete input matrix [3x2]
0009 %     Cd: discrete measurement matrix [1x3]
0010 %     Dd: 0
0011 %
0012 %     Q: discrete covariance matrix of process noise [3x3]
0013 %     R: discrete covariance matrix of measurement noise [1x1]
0014 %     x0: prior state estimate
0015 %     P0: error covariance for x0
0016 %
0017 %     in1: first input ao
0018 %     in2: second input ao
0019 %     meas: measurement data (ao)
0020 %
0021 % veronika 03-07-07
0022 %
0023
0024 % Check if this is a call for parameters (necessary forthe gui)
0025 if nargin == 1
0026     if ischar(varargin{1})
0027         if strcmp(varargin{1}, 'Params')
0028             X1 = getDefaultPlist();
0029             return
0030         end
0031     end
0032 end
0033
0034 % capture input variable names
0035 invars = {};
0036 for j=1:nargin
0037     if isa(varargin{j}, 'ao')
0038         invars = [invars cellstr(inputname(j))];
0039     end
0040 end
0041
0042 ALGONAME = mfilename;
0043 VERSION  = '$Id: ltpda_dkalman.m,v 1.4 2007/11/07 15:57:27 ingo Exp $';
0044
0045 % Get input signals

```

```

0046 Ad      = varargin{1};
0047 Bd      = varargin{2};
0048 Cd      = varargin{3};
0049 Dd      = varargin{4};
0050 Q        = varargin{5};
0051 R        = varargin{6};
0052 P0       = varargin{7};
0053 x0       = varargin{8};
0054 sigX1     = varargin{9};
0055 sigX12    = varargin{10};
0056 sigFsus   = varargin{11};
0057
0058 % Get data from analysis objects
0059 fs        = sigX12.data.fs;
0060 sigx12    = sigX12.data.y;
0061 sigx1     = sigX1.data.y;
0062 sigfsus   = sigFsus.data.y;
0063
0064
0065 u = [sigx1 sigfsus];
0066 size(sigx1);
0067 size(sigfsus);
0068 size(u);
0069
0070 for i=1:length(sigx1)
0071
0072     % kalman gain
0073     K = P0*Cd'*inv(Cd*P0*Cd'+R);
0074     % new estimation
0075     x_est = x0 + K*(sigx12(i)-Cd*x0);
0076     x1(i) = x_est(1,1);
0077     x2(i) = x_est(2,1);
0078     x3(i) = x_est(3,1);
0079     % new covariance
0080     P = (eye(3) - K*Cd)*P0;
0081     % updated state estimate
0082     x0 = Ad*x_est + Bd*u(i,:)' ;
0083     P0 = Ad*P*Ad' + Q;
0084 end
0085
0086 % History for output AOs
0087 inhist = [sigX1.hist sigX12.hist sigFsus.hist];
0088 h = history(ALGONAME, VERSION, inhist);
0089 h = set(h, 'invars', invars);
0090
0091 % Make output AOs
0092 ts = tsdata(sigX1.data.x, x1);
0093 ts = set(ts, 'name', 'x1');
0094 X1 = ao(ts, h);
0095 X1 = set(X1, 'name', 'X1');

```

```
0096
0097 ts = tsdata(sigX1.data.x, x2);
0098 ts = set(ts, 'name', 'x2');
0099 X2 = ao(ts, h);
0100 X2 = set(X2, 'name', 'X2');
0101
0102 ts = tsdata(sigX1.data.x, x3);
0103 ts = set(ts, 'name', 'x3');
0104 X3 = ao(ts, h);
0105 X3 = set(X3, 'name', 'X3');
0106
0107 %-----
0108 % get default parameter list
0109 function pl = getDefaultPlist()
0110
0111 pl = plist();
0112
0113
0114
```

[Home](#) > [classes](#) > [@tsdata](#) > [private](#) > retrieve.m

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:

- [fromxml](#) FROMXML read from part of an XML tree and create a tsdata object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
0002
```

```

0003 % RETRIEVE tsdata objects with given ids from the LTPDA repository specified by
the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE tsdata objects with given ids from the LTPDA
0009 %               repository specified by the input database connection.
0010 %
0011 % CALL:         p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:      $Id: retrieve.m,v 1.1 2007/08/31 17:40:09 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'tsdata';
0023
0024 %%%%%%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c    = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(cccccc.getNodeName), type)
0043                             obj = fromxml(cccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end
0053
0054

```

```
0055  
0056  
0057  
0058  
0059  
0060  
0061 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@miir](#) > xml.m

xml

PURPOSE

XML convert an miir object to an DOM XML object.

SYNOPSIS

function x = xml(f)

DESCRIPTION

XML convert an miir object to an DOM XML object.

%%%

DESCRIPTION: XML convert an miir object to an DOM XML object.

CALL: >> x = xml(f)

VERSION: \$Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add an MIIR object to the given node of the input XML DOM

This function is called by:

- [xmladd](#) XMLADD add an MIIR object to the given node of the input XML DOM

SOURCE CODE

```
0001 function x = xml(f)
0002 % XML convert an miir object to an DOM XML object.
0003 %
```



```
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert an miir object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(f)
0009 %
0010 % VERSION: $Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('miir');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(f, x, 'Object', x.docRootNode);
0025
0026
0027 % END
```

[Home](#) > [m](#) > [helper](#) > ltpda_fields2list.m

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

```

0001 function fnames = ltpda_fields2list(fields)
0002 % LTPDA_FIELDS2LIST splits a string containing fields seperated by ','
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_FIELDS2LIST splits a string containing fields seperated by
0007 %               ',' and returns a cell array.
0008 %
0009 % CALL:         fnames = ltpda_fields2list(fields)
0010 %
0011 % INPUTS:       string - a string seperated by ','
0012 %
0013 % OUTPUTS:      fnames - cell array
0014 %
0015 % EXAMPLE:      >> fields = 'field1, field2, field3, field4';
0016 %               >> ltpda_fields2list(fields)
0017 %               ans =
0018 %                   'field1'      ' field2'      ' field3'      ' field4'
0019 %
0020 % VERSION: $Id: ltpda_fields2list.m,v 1.5 2007/07/13 12:17:39 ingo Exp $
0021 %
0022 % HISTORY: 26-01-2007 M Hewitson
0023 %           Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026
0027 fnames = [];
0028 [f,r] = strtok(fields, ',');
0029 fnames = [fnames cellstr(f)];
0030 while ~isempty(r)
0031
0032     [f,r] = strtok(r, ',');
0033     fnames = [fnames cellstr(f)];
0034
0035 end
0036
0037
0038 % END

```

[Home](#) > [classes](#) > [@xydata](#) > [private](#) > [retrieve.m](#)

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:

- [fromxml](#) FROMXML read from part of an XML tree and create a xydata object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
0002
```

```

0003 % RETRIEVE xydata objects with given ids from the LTPDA repository specified by
the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE xydata objects with given ids from the LTPDA
0009 %               repository specified by the input database connection.
0010 %
0011 % CALL:         p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:      $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'xydata';
0023
0024 %%%%%%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(cccccc.getNodeName), type)
0043                             obj = fromxml(cccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end
0053
0054

```

```
0055  
0056  
0057  
0058  
0059  
0060  
0061 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@param](#) > xml.m

xml

PURPOSE

XML convert a param object to an DOM XML object.

SYNOPSIS

function x = xml(p)

DESCRIPTION

XML convert a param object to an DOM XML object.

%%%

DESCRIPTION: XML convert a param object to an DOM XML object.

CALL: >> x = xml(p)

VERSION: \$Id: xml.m,v 1.2 2007/09/15 04:50:19 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add a parameter object to the given node of the input XML

This function is called by:

- [xmladd](#) XMLADD add a parameter object to the given node of the input XML

SOURCE CODE

```
0001 function x = xml(p)
0002 % XML convert a param object to an DOM XML object.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert a param object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(p)
0009 %
0010 % VERSION: $Id: xml.m,v 1.2 2007/09/15 04:50:19 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('param');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(p, x, 'Object', x.docRootNode);
0025
0026 % END

```


[Home](#) > [m](#) > [helper](#) > ltpda_filescan.m

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

OUTPUTS: files - the found file names

VERSION: \$Id: ltpda_filescan.m,v 1.6 2007/11/09 14:07:39 ingo 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

```

0001 function files = ltpda_filescan(root_dir, ext)
0002 % LTPDA_FILESCAN recursively scans the given directory for files that end in
'ext'
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_FILESCAN recursively scans the given directory for files
0007 %               that end in 'ext' and returns a list of the full paths.
0008 %
0009 % CALL:        files = ltpda_filescan(root_dir, ext)
0010 %
0011 % INPUTS:      root_dir - directory
0012 %               ext      - extension of a file
0013 %
0014 % OUTPUTS:     files     - the found file names
0015 %
0016 % VERSION: $Id: ltpda_filescan.m,v 1.6 2007/11/09 14:07:39 ingo Exp $
0017 %
0018 % HISTORY: 26-01-2007 M Hewitson
0019 %           Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 files = getfiles(root_dir, ext, []);
0024
0025 %-----
0026 function ofiles = getfiles(root_dir, iext, ofiles)
0027 % Recursive function for getting file lists
0028 %
0029
0030 % mdisp('  + looking in %s', root_dir);
0031 files = dir(root_dir);
0032
0033 for j=1:length(files)
0034     f = files(j);
0035     if f.isdir
0036         if strcmp(f.name, '.')==0 && strcmp(f.name, '..')==0
0037             %mdisp('found dir %s', f.name);
0038             ofiles = getfiles([root_dir '/' f.name], iext, ofiles);
0039         end
0040     else
0041         [pathstr,name,ext,v] = fileparts(f.name);
0042         if strcmp(ext, iext)
0043             ofiles = [ofiles; cellstr([root_dir '/' f.name])];
0044         end
0045     end
0046 end
0047
0048 % END

```


[Home](#) > [classes](#) > [@zero](#) > [private](#) > retrieve.m

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:

- [fromxml](#) FROMXML read from part of an XML tree and create a zero object.

This function is called by:

SOURCE CODE

```
0001 function objs = retrieve(conn, ids)
```

```

0002
0003 % RETRIEVE zero objects with given ids from the LTPDA repository specified by
the input
0004 % database connection.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE zero objects with given ids from the LTPDA
0009 %                repository specified by the input database connection.
0010 %
0011 % CALL:         p = retrieve(conn, ids)
0012 %
0013 %
0014 %
0015 % VERSION:      $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $
0016 %
0017 % HISTORY: 30-08-07 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 type = 'zero';
0023
0024 %%%%%%%%%%%%% Get each object
0025 objs = [];
0026 for j=1:length(ids)
0027     % Get object
0028     xdoc = mysql_getXdoc(conn, ids(j));
0029     % get child
0030     obj = [];
0031     c    = xdoc.getChildNodes;
0032     for k = 1:c.getLength
0033         cc = c.item(k-1);
0034         if strcmp(char(cc.getNodeName), type)
0035             ccc = cc.getChildNodes;
0036             for l=1:ccc.getLength
0037                 cccc = ccc.item(l-1);
0038                 if strcmp(char(cccc.getNodeName), 'Object')
0039                     ccccc = cccc.getChildNodes;
0040                     for m=1:ccccc.getLength
0041                         cccccc = ccccc.item(m-1);
0042                         if strcmp(char(ccccccc.getNodeName), type)
0043                             obj = fromxml(ccccccc);
0044                         end
0045                     end
0046                 end
0047             end
0048         end
0049     end
0050
0051     objs = [objs obj];
0052 end

```

```
0053
0054
0055
0056
0057
0058
0059
0060
0061 % END
```

[Home](#) > [classes](#) > [@plist](#) > xml.m

xml

PURPOSE

XML convert a plist object to an DOM XML object.

SYNOPSIS

function x = xml(pl)

DESCRIPTION

XML convert a plist object to an DOM XML object.

%%%

DESCRIPTION: XML convert a plist object to an DOM XML object.

CALL: >> x = xml(pl)

VERSION: \$Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add a plist object to the given node of the input XML DOM

This function is called by:

- [xmladd](#) XMLADD add a plist object to the given node of the input XML DOM

SOURCE CODE

```
0001 function x = xml(pl)
0002 % XML convert a plist object to an DOM XML object.
0003 %
```

```
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert a plist object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(pl)
0009 %
0010 % VERSION: $Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('plist');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(pl, x, 'Object', x.docRootNode);
0025
0026
0027 % END
```


display

PURPOSE

DISPLAY implement terminal display for cdata object.

SYNOPSIS

function txt = display(cdatas, varargin)

DESCRIPTION

```
DISPLAY implement terminal display for cdata object.

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

DESCRIPTION: DISPLAY implement terminal display for cdata object.

VERSION: $Id: display.m,v 1.6 2008/01/08 16:39:11 ingo Exp $

HISTORY: 31-01-2007 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:

SOURCE CODE

```
0001 function txt = display(cdatas, varargin)
0002 % DISPLAY implement terminal display for cdata object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY implement terminal display for cdata object.
0007 %
0008 % VERSION: $Id: display.m,v 1.6 2008/01/08 16:39:11 ingo Exp $
0009 %
```

```

0010 % HISTORY: 31-01-2007 M Hewitson
0011 %           Creation
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014
0015 VERSION = '$Id: display.m,v 1.6 2008/01/08 16:39:11 ingo Exp $';
0016
0017 % Check if this is a call for parameters
0018 if nargin == 2
0019     if isa(cdatas, 'cdata') && ischar(varargin{1})
0020         in = char(varargin{1});
0021         if strcmp(in, 'Params')
0022             txt = plist();
0023             return
0024         elseif strcmp(in, 'Version')
0025             txt = VERSION;
0026             return
0027         end
0028     end
0029 end
0030
0031 txt = {};
0032
0033 for i=1:numel(cdatas)
0034     c = cdatas(i);
0035     banner = sprintf('----- cdata %02d -----', i);
0036     txt{end+1} = banner;
0037
0038     txt{end+1} = ' ';
0039
0040     y      = get(c, 'y');
0041     xinfo = whos('y');
0042
0043     txt{end+1} = sprintf('  name: %s', get(c, 'name'));
0044     txt{end+1} = sprintf('  y:  [%dx%d], %s', xinfo.size(1), xinfo.size(2), xinfo.
class);
0045     x = get(c, 'x');
0046     tstr = '';
0047     for j=1:length(x)
0048         if j < 10
0049             tstr = [tstr ' ' char(x{j})];
0050         else
0051             tstr = [deblank(tstr) '...'];
0052             break;
0053         end
0054     end
0055
0056     txt{end+1} = ['  x: ' tstr];
0057
0058     banner_end(1:length(banner)) = '-';
0059     txt{end+1} = banner_end;
0060

```

```
0061    txt{end+1} = ' ';  
0062    txt{end+1} = ' ';  
0063 end  
0064  
0065 if nargout == 0  
0066     for ii=1:length(txt)  
0067         disp(txt{ii});  
0068     end  
0069 end  
0070  
0071 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [helper](#) > ltpda_fq2ri.m

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

```
0001 function [G,ri]= ltpda_fq2ri(f0, Q)
0002
0003 % LTPDA_FQ2RI Convert frequency/Q pole/zero representation into real
0004 % and imaginary conjugate pairs.
0005 %
0006 % [G,ri]= ltpda_fq2ri(f0, Q)
0007 %
0008 % M Hewitson 26-01-07
```

```
0009 %
0010 % $Id: ltpda_fq2ri.m,v 1.4 2007/01/26 14:03:24 hewitson Exp $
0011 %
0012
0013 if(nargin==0)
0014     disp('usage: [G,ri]= ltpda_fq2ri(f0, Q)')
0015     return
0016 end
0017
0018 % a = Q - 0.5 ;
0019 % re = 2*pi*f0 / sqrt(1+tan(a)^2);
0020 % im = re * tan(a);
0021
0022 w0 = 2*pi*f0;
0023 re = -w0/(2*Q);
0024 im = w0*sqrt(4*Q*Q-1)/(2*Q);
0025 G = 1/(w0*w0);
0026 ri = complex(re,im);
0027
0028 % END
```

[Home](#) > [classes](#) > [@pole](#) > rp2iir.m

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 1.4 2008/01/11 13:42:41 ingo 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

```

0001 function [a,b] = rp2iir(p, fs)
0002 % RP2IIR Return a,b coefficients for a real pole designed using the bilinear
transform.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: RP2IIR Return a,b coefficients for a real pole designed using
0007 %               the bilinear transform.
0008 %
0009 % CALL:         filt = rpole(p, fs)
0010 %
0011 % REMARK:       This is just a helper function. This function should only be
0012 %               called from class functions.
0013 %
0014 % INPUT:        p - pole object
0015 %               fs - the sample rate for the filter
0016 %
0017 % VERSION:      $Id: rp2iir.m,v 1.4 2008/01/11 13:42:41 ingo Exp $
0018 %
0019 % HISTORY:      03-03-07 M Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: rp2iir.m,v 1.4 2008/01/11 13:42:41 ingo Exp $';
0025
0026 %%%% 'Params' && 'Version' Call %%%%
0027 if nargin == 2
0028     if isa(p, 'pole') && ischar(fs)
0029         in = fs;
0030         if strcmp(in, 'Params')
0031             a = plist();
0032             return
0033         elseif strcmp(in, 'Version')
0034             a = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 f0 = get(p, 'f');
0041 w0 = f0*2*pi;
0042 a(1) = w0 / (2*fs + w0);
0043 a(2) = a(1);
0044 b(1) = 1;
0045 b(2) = (w0-2*fs) / (w0+2*fs);
0046
0047 % END

```

[Home](#) > [classes](#) > [@pole](#) > xml.m

xml

PURPOSE

XML convert a pole object to an DOM XML object.

SYNOPSIS

function x = xml(p)

DESCRIPTION

XML convert a pole object to an DOM XML object.

%%%

DESCRIPTION: XML convert a pole object to an DOM XML object.

CALL: >> x = xml(p)

VERSION: \$Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add a pole object to the given node of the input XML DOM

This function is called by:

- [xmladd](#) XMLADD add a pole object to the given node of the input XML DOM

SOURCE CODE

```
0001 function x = xml(p)
0002 % XML convert a pole object to an DOM XML object.
0003 %
```



```
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert a pole object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(p)
0009 %
0010 % VERSION: $Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('pole');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(p, x, 'Object', x.docRootNode);
0025
0026
0027 % END
```

[Home](#) > [classes](#) > [@fsdata](#) > display.m

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 1.7 2008/01/08 16:39:11 ingo 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:

SOURCE CODE

```
0001 function txt = display(fsdatas, varargin)
0002 % DISPLAY implement terminal display for fsdata object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY implement terminal display for fsdata object.
0007 %
0008 % VERSION: $Id: display.m,v 1.7 2008/01/08 16:39:11 ingo Exp $
```

```

0009 %
0010 % HISTORY: 31-01-2007 M Hewitson
0011 %           Creation
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014
0015 VERSION = '$Id: display.m,v 1.7 2008/01/08 16:39:11 ingo Exp $';
0016
0017 % Check if this is a call for parameters
0018 if nargin == 2
0019     if isa(fsdatas, 'fsdata') && ischar(varargin{1})
0020         in = char(varargin{1});
0021         if strcmp(in, 'Params')
0022             txt = plist();
0023             return
0024         elseif strcmp(in, 'Version')
0025             txt = VERSION;
0026             return
0027         end
0028     end
0029 end
0030
0031 txt = {};
0032
0033 for i=1:numel(fsdatas)
0034     fsd = fsdatas(i);
0035
0036     banner_start = sprintf('----- fsdata %02d -----', i);
0037     txt{end+1} = banner_start;
0038     txt{end+1} = ' ';
0039
0040     xdata = get(fsd, 'x');
0041     ydata = get(fsd, 'y');
0042     xinfo = whos('xdata');
0043     yinfo = whos('ydata');
0044
0045     txt{end+1} = sprintf('  name: %s', get(fsd, 'name'));
0046     txt{end+1} = sprintf('    fs: %g', get(fsd, 'fs'));
0047     txt{end+1} = sprintf('    x: [%d %d], %s', xinfo.size(1), xinfo.size(2),
xinfo.class);
0048     txt{end+1} = sprintf('    y: [%d %d], %s', yinfo.size(1), yinfo.size(2),
yinfo.class);
0049     txt{end+1} = sprintf('xunits: %s', get(fsd, 'xunits'));
0050     txt{end+1} = sprintf('yunits: %s', get(fsd, 'yunits'));
0051
0052     banner_end(1:length(banner_start)) = '-';
0053     txt{end+1} = banner_end;
0054
0055     txt{end+1} = ' ';
0056     txt{end+1} = ' ';
0057 end

```

```
0058
0059 if nargout == 0
0060     for ii=1:length(txt)
0061         disp(txt{ii});
0062     end
0063 end
0064
0065 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [helper](#) > ltpda_fq2ri2.m

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

```
0001 function ri= ltpda_fq2ri2(f0, Q)
0002
0003 % LTPDA_FQ2RI2 Convert frequency/Q pole/zero representation into real
0004 % and imaginary conjugate pairs. Returns [ri conj(ri)]
0005 %
0006 % ri= ltpda_fq2ri2(f0, Q)
0007 %
0008 % M Hewitson 26-01-07
```

```
0009 %
0010 %
0011 % $Id: ltpda_fq2ri2.m,v 1.6 2007/07/11 14:06:39 anneke Exp $
0012 %
0013
0014 if(nargin==0)
0015     disp('usage: ri = fq2ri2(f0, Q)')
0016     return
0017 elseif(nargin==1)
0018     ri = (1/(2*pi*f0));
0019 elseif(nargin==2)
0020     if Q <= 0.5
0021         disp('working on Q < 0.5') %ri =1/(2*pi*f0);
0022     else
0023         w0 = 2*pi*f0;
0024         G = 1/(w0*w0);
0025         re = -w0/(2*Q);
0026         im = w0*sqrt(4*Q*Q-1)/(2*Q);
0027         tri = complex(re,im);
0028         ri = G.*[tri conj(tri)];
0029     end
0030 end
0031
0032 % END
```

[Home](#) > [m](#) > [helper](#) > rstruct.m

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

```
0001 function s = rstruct(obj)
0002 % RSTRUCT recursively converts an object into a structure. This is the same
0003 % behaviour as MATLAB's struct(obj) except that it recursively converts all
0004 % sub-objects into structures as well.
0005 %
0006 % M Hewitson 02-06-07
0007 %
0008 % $Id: rstruct.m,v 1.1 2007/09/18 17:44:36 hewitson Exp $
0009 %
0010
0011 s = struct(obj);
0012 names = fieldnames(s);
0013 fields = struct2cell(s);
0014
```

```
0015 for j=1:length(names)
0016
0017     f = fields{j};
0018     if isobject(f)
0019         try
0020             cmd = sprintf('s.%s = rstruct(f);', names{j});
0021             eval(cmd);
0022         end
0023     end
0024 end
```


[Home](#) > [classes](#) > [@provenance](#) > xml.m

xml

PURPOSE

XML convert a provenance object to an DOM XML object.

SYNOPSIS

function x = xml(s)

DESCRIPTION

XML convert a provenance object to an DOM XML object.

%%%

DESCRIPTION: XML convert a provenance object to an DOM XML object.

CALL: >> x = xml(s)

VERSION: \$Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD adds specwin object to the given node of the xml input

This function is called by:

- [xmladd](#) XMLADD adds specwin object to the given node of the xml input

SOURCE CODE

```
0001 function x = xml(s)
0002 % XML convert a provenance object to an DOM XML object.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert a provenance object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(s)
0009 %
0010 % VERSION: $Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('provenance');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(s, x, 'Object', x.docRootNode);
0025
0026
0027 % END

```

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 1.9 2008/01/08 16:39:11 ingo Exp \$

HISTORY: 30-01-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a param object into a string.
- [display](#) DISPLAY 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.

SUBFUNCTIONS

- `function new_txt = single_cell(txt_field)`

SOURCE CODE

```

0001 function txt = display(hists, varargin)
0002 % DISPLAY implement terminal display for history object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY implement terminal display for history object.
0007 %
0008 % CALL:          txt = display(history)
0009 %
0010 % INPUT:         history - history object
0011 %
0012 % OUTPUT:        txt      - cell array with strings to display the history object
0013 %
0014 % VERSION:       $Id: display.m,v 1.9 2008/01/08 16:39:11 ingo Exp $
0015 %
0016 % HISTORY:       30-01-2007 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: display.m,v 1.9 2008/01/08 16:39:11 ingo Exp $';
0022
0023 % Check if this is a call for parameters
0024 if nargin == 2
0025     if isa(hists, 'history') && ischar(varargin{1})
0026         in = char(varargin{1});
0027         if strcmp(in, 'Params')
0028             txt = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             txt = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 txt = {};
0038
0039 for i=1:numel(hists)
0040     h = hists(i);
0041     banner = sprintf('----- %s / %s / %d -----', get(h, 'name'), ...
0042                     get(h, 'version'), ...
0043                     get(h, 'n'));
0044     txt{end+1} = banner;
0045
0046     txt{end+1} = ' ';

```

```

0047     txt{end+1} = sprintf('created: %s', char(h.created));
0048
0049     % show input vars
0050     txt{end+1} = ['Inputs: ' get(h, 'invars')];
0051
0052     % show parameter list
0053     txt{end+1} = 'Parameters';
0054     txt{end+1} = '-----';
0055     pl = get(h, 'plist');
0056     if ~isempty(pl)
0057         txt{end+1} = display(pl);
0058     end
0059
0060     txt{end+1} = ' ';
0061
0062     txt{end+1} = 'Histories';
0063     txt{end+1} = '-----';
0064
0065     % show input histories
0066     inhists = get(h, 'inhists');
0067     nhists = length(inhists);
0068     txt{end+1} = sprintf('\n input histories: %d', nhists);
0069     if nhists > 0
0070         txt{end+1} = display(inhists);
0071     end
0072
0073     txt{end+1} = ' ';
0074
0075     bannerEnd = [];
0076     while length(bannerEnd) < length(banner)
0077         bannerEnd = [bannerEnd '-'];
0078     end
0079     txt{end+1} = bannerEnd;
0080
0081     txt{end+1} = ' ';
0082     txt{end+1} = ' ';
0083 end
0084
0085 txt = single_cell(txt);
0086
0087 if nargout == 0
0088     for ii=1:length(txt)
0089         disp(txt{ii});
0090     end
0091 end
0092
0093 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0094
0095 function new_txt = single_cell(txt_field)
0096
0097 new_txt = {};

```

```
0098 for ii=1:length(txt_field)
0099     if iscell(txt_field{ii})
0100         hh = single_cell(txt_field{ii});
0101         new_txt(end+1:end+length(hh)) = hh(1:end);
0102     else
0103         new_txt{end+1} = txt_field{ii};
0104     end
0105 end
0106
0107 % END
```

[Home](#) > [m](#) > [helper](#) > ltpda_funchash.m

ltpda_funchash

PURPOSE

LTPDA_FUNCHASH compute MD5 hash of an LTPDA function.

SYNOPSIS

function h = ltpda_funchash(fcnnname)

DESCRIPTION

LTPDA_FUNCHASH compute MD5 hash of an LTPDA function.

%%%

DESCRIPTION: LTPDA_FUNCHASH compute MD5 hash of an LTPDA function.

CALL: h = ltpda_funchash(fcnnname)

INPUTS: fcnnname - 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

```

0001 function h = ltpda_funchash(fcname)
0002
0003 % LTPDA_FUNCHASH compute MD5 hash of an LTPDA function.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: LTPDA_FUNCHASH compute MD5 hash of an LTPDA function.
0008 %
0009 % CALL:          h = ltpda_funchash(fcname)
0010 %
0011 % INPUTS:        fcname - An LTPDA function name
0012 %
0013 % OUTPUTS:       h - the hash string
0014 %
0015 % PARAMETERS: None.
0016 %
0017 % EXAMPLES:
0018 %
0019 %     >> h = ltpda_funchash('ao');
0020 %
0021 % VERSION:       $Id: ltpda_funchash.m,v 1.1 2007/11/21 13:59:29 hewitson Exp $
0022 %
0023 % HISTORY: 21-11-07 M Hewitson
0024 %           Creation.
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0027
0028 ALGONAME = mfilename;
0029 VERSION  = '$Id: ltpda_funchash.m,v 1.1 2007/11/21 13:59:29 hewitson Exp $';
0030
0031 % Get filename
0032 s = which(fcname);
0033
0034 % Read in file
0035 mfile = textread(s, '%s', 'delimiter', '\n', 'whitespace', '');
0036
0037 % hash this
0038 h = ltpda_hash(char(mfile), 'MD5');
0039

```


[Home](#) > [classes](#) > [@zero](#) > rz2iir.m

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 1.5 2008/01/11 13:42:41 ingo 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

```

0001 function [a,b] = rz2iir(z, fs)
0002 % RZ2IIR Return a,b IIR filter coefficients for a real zero designed using the
0003 % bilinear transform.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: RZ2IIR Return a,b IIR filter coefficients for a real zero
0007 %               designed using the bilinear transform.
0008 %
0009 % CALL:        [a,b] = rz2iir(z, fs)
0010 %
0011 % REMARK:      This is just a helper function. This function should only be
0012 %               called from class functions.
0013 %
0014 % INPUT:       z - zero object
0015 %               fs - the sample rate for the filter
0016 %
0017 % VERSION:     $Id: rz2iir.m,v 1.5 2008/01/11 13:42:41 ingo Exp $
0018 %
0019 % HISTORY:     18-02-2003 Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: rz2iir.m,v 1.5 2008/01/11 13:42:41 ingo Exp $';
0025
0026 %%%% 'Params' && 'Version' Call %%%%
0027 if nargin == 2
0028     if isa(z, 'zero') && ischar(fs)
0029         in = fs;
0030         if strcmp(in, 'Params')
0031             a = plist();
0032             return
0033         elseif strcmp(in, 'Version')
0034             a = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 f0 = get(z, 'f');
0041 w0 = f0*2*pi;
0042
0043 a(1) = (2*fs + w0) / w0;
0044 a(2) = (-2*fs + w0) / w0;
0045
0046 b(1) = 1;
0047 b(2) = 1;

```

[Home](#) > [classes](#) > [@pzmodel](#) > xml.m

xml

PURPOSE

XML convert a pzmodel object to an DOM XML object.

SYNOPSIS

function x = xml(pzm)

DESCRIPTION

XML convert a pzmodel object to an DOM XML object.

%%%

DESCRIPTION: XML convert a pzmodel object to an DOM XML object.

CALL: >> x = xml(pzm)

VERSION: \$Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add a pzmodel object to the given node of the input XML DOM

This function is called by:

- [xmladd](#) XMLADD add a pzmodel object to the given node of the input XML DOM

SOURCE CODE

```
0001 function x = xml(pzm)
0002 % XML convert a pzmodel object to an DOM XML object.
0003 %
```

```
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert a pzmodel object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(pzm)
0009 %
0010 % VERSION: $Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('pzmodel');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(pzm, x, 'Object', x.docRootNode);
0025
0026
0027 % END
```

[Home](#) > [classes](#) > [@mfir](#) > display.m

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 1.7 2008/01/08 16:39:11 ingo 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.

SOURCE CODE

```

0001 function txt = display(fs, varargin)
0002 % DISPLAY overloads display functionality for mfir objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY overloads display functionality for mfir objects.
0007 %
0008 % CALL:          txt = display(mfir)
0009 %
0010 % INPUT:         mfir - FIR filter object
0011 %
0012 % OUTPUT:        txt - cell array with strings to display the mfir object
0013 %
0014 % VERSION:       $Id: display.m,v 1.7 2008/01/08 16:39:11 ingo Exp $
0015 %
0016 % HISTORY:       11-01-2007 M Hewitson
0017 %                  Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: display.m,v 1.7 2008/01/08 16:39:11 ingo Exp $';
0022
0023 % Check if this is a call for parameters
0024 if nargin == 2
0025     if isa(fs, 'mfir') && ischar(varargin{1})
0026         in = char(varargin{1});
0027         if strcmp(in, 'Params')
0028             txt = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             txt = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 txt = {};
0038
0039 for i=1:numel(fs)
0040     f = fs(i);
0041     banner_start = sprintf('----- %s / %s -----', ...
0042                             get(f, 'name'), ...
0043                             get(f, 'version'));
0044     txt{end+1} = banner_start;
0045
0046     txt{end+1} = ' ';
0047     txt{end+1} = sprintf('created: %s', char(f.created));
0048     txt{end+1} = sprintf('version: %s', f.version);
0049     txt{end+1} = sprintf('    name: %s', f.name);
0050     txt{end+1} = ['    fs: ' num2str(f.fs)];
0051     txt{end+1} = ['    ntaps: ' num2str(f.ntaps)];

```

```
0052 txt{end+1} = ['      a: ' num2str(f.a)];
0053 txt{end+1} = ['    gain: ' num2str(f.gain)];
0054 txt{end+1} = ['histout: ' num2str(f.histout)];
0055
0056 txt{end+1} = sprintf('Parameters:');
0057 txt_plist = display(f.plist);
0058
0059 for ii = 1:length(txt_plist)
0060     txt{end+1} = txt_plist{ii};
0061 end
0062
0063 txt{end+1} = ' ';
0064
0065 banner_end(1:length(banner_start)) = '-';
0066 txt{end+1} = banner_end;
0067
0068 txt{end+1} = ' ';
0069 txt{end+1} = ' ';
0070 end
0071
0072 if nargout == 0
0073     for ii=1:length(txt)
0074         disp(txt{ii});
0075     end
0076 end
0077
0078
0079 % END
```

[Home](#) > [m](#) > [helper](#) > ltpda_getmax.m

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 of 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

```
0001 function [index, max] = ltpda_getmax(a)
0002 % LTPDA_GETMAX get the index and value of the maximum value in an array.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```



```

0005 %
0006 % DESCRIPTION: LTPDA_GETMAX get the index and value of the maximum value in an
array
0007 %
0008 % CALL:          [index, max] = ltpda_getmax(a)
0009 %
0010 % INPUTS:        a          - vector of double
0011 %
0012 % OUTPUTS:        index - index of the maximum
0013 %                  max   - value of the maximum
0014 %
0015 % VERSION: $Id: ltpda_getmax.m,v 1.5 2007/07/13 12:17:39 ingo Exp $
0016 %
0017 % HISTORY: 26-01-2007 M Hewitson
0018 %              Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 max = a(1);
0023 index = 1;
0024
0025 for k=1:length(a)
0026     if(a(k) > max)
0027         max = a(k);
0028         index = k;
0029     end
0030 end
0031
0032
0033 % END

```

[Home](#) > [classes](#) > [@specwin](#) > xml.m

xml

PURPOSE

XML convert a specwin object to an DOM XML object.

SYNOPSIS

function x = xml(s)

DESCRIPTION

XML convert a specwin object to an DOM XML object.

%%%

DESCRIPTION: XML convert a specwin object to an DOM XML object.

CALL: >> x = xml(s)

VERSION: \$Id: xml.m,v 1.2 2007/09/15 04:50:19 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD adds specwin object to the given node of the xml input

This function is called by:

SOURCE CODE

```
0001 function x = xml(s)
0002 % XML convert a specwin object to an DOM XML object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
```

```
0006 % DESCRIPTION: XML convert a specwin object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(s)
0009 %
0010 % VERSION: $Id: xml.m,v 1.2 2007/09/15 04:50:19 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('specwin');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(s, x, 'Object', x.docRootNode);
0025
0026
0027 % END
```

[Home](#) > [classes](#) > [@miir](#) > display.m

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 1.10 2008/01/08 16:39:11 ingo 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.

SOURCE CODE

```

0001 function txt = display(fs, varargin)
0002 % DISPLAY overloads display functionality for miir objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY overloads display functionality for miir objects.
0007 %
0008 % CALL:          txt = display(miir)
0009 %
0010 % INPUT:         miir - IIR filter object
0011 %
0012 % OUTPUT:        txt - cell array with strings to display the miir object
0013 %
0014 % VERSION:       $Id: display.m,v 1.10 2008/01/08 16:39:11 ingo Exp $
0015 %
0016 % HISTORY:       11-01-2007 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: display.m,v 1.10 2008/01/08 16:39:11 ingo Exp $';
0022
0023 % Check if this is a call for parameters
0024 if nargin == 2
0025     if isa(fs, 'miir') && ischar(varargin{1})
0026         in = char(varargin{1});
0027         if strcmp(in, 'Params')
0028             txt = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             txt = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 txt = {};
0038
0039 for i=1:numel(fs)
0040     f = fs(i);
0041     banner_start = sprintf('----- %s / %s -----', get(f, 'name'), get
(f, 'version'));
0042     txt{end+1} = banner_start;
0043
0044     txt{end+1} = ' ';
0045     txt{end+1} = sprintf('created: %s', char(f.created));
0046     txt{end+1} = sprintf('version: %s', f.version);
0047     txt{end+1} = sprintf('    name: %s', f.name);
0048     txt{end+1} = ['    fs: ' num2str(f.fs)];
0049     txt{end+1} = ['    ntaps: ' num2str(f.ntaps)];
0050     txt{end+1} = ['    a: ' num2str(f.a)];

```

```
0051     txt{end+1} = ['      b: ' num2str(f.b)];
0052     txt{end+1} = ['    gain: ' num2str(f.gain)];
0053     txt{end+1} = ['  histin: ' num2str(f.histin)];
0054     txt{end+1} = ['histout: ' num2str(f.histout)];
0055
0056     txt{end+1} = sprintf('Parameters:');
0057     txt_plist = display(f.plist);
0058
0059     for ii = 1:length(txt_plist)
0060         txt{end+1} = txt_plist{ii};
0061     end
0062
0063     txt{end+1} = ' ';
0064
0065     banner_end(1:length(banner_start)) = '-';
0066     txt{end+1} = banner_end;
0067
0068     txt{end+1} = ' ';
0069     txt{end+1} = ' ';
0070 end
0071
0072 if nargout == 0
0073     for ii=1:length(txt)
0074         disp(txt{ii});
0075     end
0076 end
0077
0078
0079 % END
```

[Home](#) > [m](#) > [timetools](#) > ltpda_gps2utc.m

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

```
0001 function UTC_time=ltpda_gps2utc(GPS_time)
```

```

0002 % LTPDA_GPS2UTC converts GPS seconds to UTC time.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_GPS2UTC converts GPS seconds to UTC time.
0007 %
0008 % CALL:          UTC_time=ltpda_GPS2UTC(GPS_time)
0009 %
0010 % FORMAT:        UTC time format: 'yyy-mm-dd- HH:MM:SS'
0011 %                GPS time format: Seconds since 6. January 1980
0012 %
0013 %   EXAMPLES:    GPS_time=ltpda_GPS2UTC(711129613)
0014 %                GPS_time='2002-07-19 16:00:00'
0015 %
0016 % VERSION:       $Id: ltpda_gps2utc.m,v 1.4 2007/07/27 09:31:22 ingo Exp $
0017 %
0018 % HISTORY:        xx-xx-xxxx Karsten Koetter
0019 %                Creation
0020 %                24-01-2007 M Hewitson.
0021 %                Maintained
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 GPS_Epoch=datetime('01-06-1980 00:00:00')*86400;
0026 NUM_time=GPS_Epoch+GPS_time-14;
0027
0028 UTC_time=datestr(NUM_time/86400,31);

```


[Home](#) > [classes](#) > [@cdata](#) > save.m

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 1.1 2007/08/29 18:45:00 hewitson Exp \$

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

```
>> pl = save(cdata, 'Params')
```

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE a cdata object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for cdata objects.
0008 %
0009 % CALL:          Save a cdata object as a .mat file.
0010 %                >> save(d, 'blah.mat')
0011 %                Save a cdata object as an XML file.
0012 %                >> save(d, 'blah.xml')
0013 %                Save a cdata object to file specified in plist.
0014 %                >> save(d, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:      $Id: save.m,v 1.1 2007/08/29 18:45:00 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(cdata, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'cdata') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041 ALGONAME = mfilename;
0042 VERSION  = '$Id: save.m,v 1.1 2007/08/29 18:45:00 hewitson Exp $';

```

```
0043
0044 if nargin ~= 2
0045     help mfilename;
0046     error('### incorrect inputs');
0047 end
0048
0049 % get object
0050 obj = varargin{1};
0051 if isa(varargin{2}, 'plist')
0052     pl = varargin{2};
0053 elseif ischar(varargin{2})
0054     pl = plist(param('filename', varargin{2}));
0055 else
0056     help mfilename;
0057     error('### incorrect inputs');
0058 end
0059
0060 % Save object
0061 ltpda_saveobj(obj, pl);
0062
0063 % Get default params
0064 function plo = getDefaultPL()
0065
0066 disp('* creating default plist...');
0067 plo = plist();
0068 plo = append(plo, param('filename', 'object.xml'));
0069 disp('* done. ');
0070
0071
0072 % END
```

[Home](#) > [classes](#) > [@time](#) > xml.m

xml

PURPOSE

XML convert a time object to an DOM XML object.

SYNOPSIS

function x = xml(tf)

DESCRIPTION

XML convert a time object to an DOM XML object.

%%%

DESCRIPTION: XML convert a time object to an DOM XML object.

CALL: >> x = xml(tf)

VERSION: \$Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add a time object to the given node of the input XML DOM

This function is called by:

- [xmladd](#) XMLADD add a time object to the given node of the input XML DOM

SOURCE CODE

```
0001 function x = xml(tf)
0002 % XML convert a time object to an DOM XML object.
0003 %
```

```
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert a time object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(tf)
0009 %
0010 % VERSION: $Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('time');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(tf, x, 'Object', x.docRootNode);
0025
0026
0027 % END
```

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 1.20 2007/12/11 17:27:09 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:

SOURCE CODE

```
0001 function txt = display(p, varargin)
0002 % DISPLAY display a parameter
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY display a parameter
0007 %
0008 % VERSION: $Id: display.m,v 1.20 2007/12/11 17:27:09 ingo Exp $
```

```

0009 %
0010 % HISTORY: 30-01-2007 M Hewitson
0011 %           Creation
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014
0015 VERSION = '$Id: display.m,v 1.20 2007/12/11 17:27:09 ingo Exp $';
0016
0017 % Check if this is a call for parameters
0018 if nargin == 2
0019     if isa(p, 'param') && ischar(varargin{1})
0020         in = char(varargin{1});
0021         if strcmp(in, 'Params')
0022             txt = plist();
0023             return
0024         elseif strcmp(in, 'Version')
0025             txt = VERSION;
0026             return
0027         end
0028     end
0029 end
0030
0031 txt = {};
0032
0033 for i=1:numel(p)
0034     banner = sprintf('---- param %d ----', i);
0035     txt{end+1} = banner;
0036
0037     % get key and value
0038     name = get(p(i), 'key');
0039     v     = get(p(i), 'val');
0040
0041     txt{end+1} = ['key: ' name];
0042
0043     % convert value for printing
0044
0045
0046     %%%% Character %%%%
0047     if ischar(v)
0048         if length(v) > 45
0049             txt{end+1} = ['val: '' v(1:45) ' ...''];
0050         else
0051             txt{end+1} = ['val: '' v '''];
0052         end
0053
0054     %%%% Logical %%%%
0055     elseif islogical(v)
0056         if v == true
0057             txt{end+1} = 'val: true';
0058         else
0059             txt{end+1} = 'val: false';
0060     end

```

```

0061
0062 %%%% Numeric %%%%
0063 elseif isnumeric(v)
0064     sval = '';
0065     if size(v,1) == 1 || size(v,2) == 1
0066         for j=1:min(length(v), 10)
0067             sval = [sval sprintf('%g ', v(j))];
0068         end
0069         if length(v) > 10
0070             sval = [sval '...'];
0071         end
0072     elseif isempty(v)
0073         sval = '[]';
0074     else
0075         sval = sprintf('Matrix [%dx%d]', size(v,1), size(v,2));
0076     end
0077     txt{end+1} = ['val: ' sval];
0078
0079 %%%% Specwin Object %%%%
0080 elseif isa(v, 'specwin');
0081     txt{end+1} = ['val: ' v.name];
0082     txt{end+1} = ['    alpha    = ' num2str(v.alpha)];
0083     txt{end+1} = ['    psll     = ' num2str(v.psll)];
0084     txt{end+1} = ['    rov      = ' num2str(v.rov)];
0085     txt{end+1} = ['    nenbw    = ' num2str(v.nenbw)];
0086     txt{end+1} = ['    w3db     = ' num2str(v.w3db)];
0087     txt{end+1} = ['    flatness = ' num2str(v.flatness)];
0088
0089 %%%% Provenance Object %%%%
0090 elseif isa(v, 'provenance')
0091     txt{end+1} = ['val: ' v.creator '@' v.hostname ' (' v.ip ')'];
0092
0093 %%%% miir and mfir Object %%%%
0094 elseif isa(v, 'miir') || isa(v, 'mfir')
0095     txt{end+1} = ['val: ' char(v)];
0096
0097 %%%% time, timespan and timeformat Object %%%%
0098 elseif isa(v, 'time') || isa(v, 'timespan') || isa(v, 'timeformat')
0099     txt{end+1} = ['val: ' char(v)];
0100
0101 %%%% ao %%%%
0102 %%%% cdata, fsdata, tsdata, xydata %%%%
0103 %%%% plist, param %%%%
0104 %%%% history %%%%
0105 elseif isa(v, 'ao') || isa(v, 'fsdata') || isa(v, 'tsdata') || ...
0106     isa(v, 'cdata') || isa(v, 'xydata') || isa(v, 'plist') || ...
0107     isa(v, 'param') || isa(v, 'history')
0108     vinfo = whos('v');
0109     txt{end+1} = ['val: ' vinfo.class '-object'];
0110
0111 %%%% pole, zero and pole/zero model Object %%%%
0112 elseif isa(v, 'pole') || isa(v, 'zero') || isa(v, 'pzmodel')

```



```

0113     sval = '';
0114     for j=1:length(v)
0115         sval = [sval ' ' char(v(j))];
0116     end
0117     txt{end+1} = ['val: ' sval];
0118
0119     %%%% Structures %%%%
0120     elseif isstruct(v)
0121
0122         fields = fieldnames(v);
0123         txt{end+1} = 'val: structure';
0124         for ii=1:length(fields)
0125             field = fields{ii};
0126             if ischar(v.(field))
0127                 txt{end+1} = [' ' field ': ' v.(field) ''];
0128             elseif isnumeric(v.(field))
0129                 txt{end+1} = [' ' field ': ' ltpda_mat2str(v.(field))];
0130             elseif isobject(v.(field))
0131                 txt{end+1} = [' ' field ': ' class(v.(field)) '-object'];
0132             else
0133                 error('### Do not use a [%s] in a cell array', class(v));
0134             end
0135         end
0136
0137     %%%% Cell %%%%
0138     elseif iscell(v)
0139         sval = '';
0140         for ii = 1:length(v)
0141             if ischar(v{ii})
0142                 sval = [sval ' ' v{ii} ''];
0143             elseif isnumeric(v{ii})
0144                 sval = [sval '[' ltpda_mat2str(v{ii}) ' '];
0145             elseif isobject(v{ii})
0146                 sval = [sval ' ' class(v{ii}) ''];
0147             else
0148                 error('### Do not use a [%s] in a cell array', class(v));
0149             end
0150         end
0151         txt{end+1} = ['val: ' sval];
0152
0153     %%%% JAVA %%%%
0154     elseif isjava(v)
0155         txt{end+1} = ['val: ' class(v)];
0156
0157     else
0158         error('### parameter has unknown type')
0159     end
0160
0161     banner_end(1:length(banner)) = '-';
0162     txt{end+1} = banner_end;
0163
0164 end

```

```
0165
0166 if nargout == 0
0167     for ii=1:length(txt)
0168         disp(txt{ii});
0169     end
0170 end
0171
0172
0173
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [helper](#) > ltpda_hash.m

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 following 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

```
0001 function h = ltpda_hash(inp, meth)
0002 % LTPDA_HASH - Convert an input variable into a message digest using any of
0003 %               several common hash algorithms
0004 %
0005 %%%%%%%%%%%
0006 %
0007 % LTPDA_HASH - Convert an input variable into a message digest using any of
0008 %               several common hash algorithms
0009 %
0010 % USAGE: h = hash(inp, 'meth')
0011 %
0012 % inp  = input variable, of any of the following classes:
0013 %       char, uint8, logical, double, single, int8, uint8,
0014 %       int16, uint16, int32, uint32, int64, uint64
0015 % h    = hash digest output, in hexadecimal notation
0016 % meth = hash algorithm, which is one of the following:
0017 %       MD2, MD5, SHA-1, SHA-256, SHA-384, or SHA-512
0018 %
```

```

0019 % NOTES: (1) If the input is a string or uint8 variable, it is hashed
0020 %           as usual for a byte stream. Other classes are converted into
0021 %           their byte-stream values. In other words, the hash of the
0022 %           following will be identical:
0023 %           'abc'
0024 %           uint8('abc')
0025 %           char([97 98 99])
0026 %           The hash of the following will be different from the above,
0027 %           because class "double" uses eight byte elements:
0028 %           double('abc')
0029 %           [97 98 99]
0030 %           You can avoid this issue by making sure that your inputs
0031 %           are strings or uint8 arrays.
0032 %           (2) The name of the hash algorithm may be specified in lowercase
0033 %           and/or without the hyphen, if desired. For example,
0034 %           h=hash('my text to hash','sha256');
0035 %           (3) Carefully tested, but no warranty. Use at your own risk.
0036 %           (4) Michael Kleder, Nov 2005
0037 %           (5) This is a direct copy of the code from Michael Kleder. Only
0038 %           the function name was changed to bring it in line with the
0039 %           LTPDA naming convention. (M Hewitson 15-09-07)
0040 %
0041 % EXAMPLE:
0042 %
0043 % algs={'MD2','MD5','SHA-1','SHA-256','SHA-384','SHA-512'};
0044 % for n=1:6
0045 %     h=hash('my sample text',algs{n});
0046 %     disp([algs{n} ' (' num2str(length(h)*4) ' bits):'])
0047 %     disp(h)
0048 % end
0049 %
0050 % $Id: ltpda_hash.m,v 1.1 2007/09/15 04:54:44 hewitson Exp $
0051 %
0052 %
0053
0054 inp=inp(:);
0055 % convert strings and logicals into uint8 format
0056 if ischar(inp) || islogical(inp)
0057     inp=uint8(inp);
0058 else % convert everything else into uint8 format without loss of data
0059     inp=typecast(inp,'uint8');
0060 end
0061
0062 % verify hash method, with some syntactical forgiveness:
0063 meth=upper(meth);
0064 switch meth
0065     case 'SHA1'
0066         meth='SHA-1';
0067     case 'SHA256'
0068         meth='SHA-256';

```

```
0069     case 'SHA384'
0070         meth='SHA-384';
0071     case 'SHA512'
0072         meth='SHA-512';
0073     otherwise
0074 end
0075 algs={'MD2','MD5','SHA-1','SHA-256','SHA-384','SHA-512'};
0076 if isempty(strmatch(meth,algs,'exact'))
0077     error(['Hash algorithm must be ' ...
0078         'MD2, MD5, SHA-1, SHA-256, SHA-384, or SHA-512']);
0079 end
0080
0081 % create hash
0082 x=java.security.MessageDigest.getInstance(meth);
0083 x.update(inp);
0084 h=typecast(x.digest,'uint8');
0085 h=dec2hex(h)';
0086 if(size(h,1))==1 % remote possibility: all hash bytes < 128, so pad:
0087     h=[repmat('0',[1 size(h,2)]);h];
0088 end
0089 h=lower(h(:)');
0090 clear x
0091 return
```

[Home](#) > [classes](#) > [@fsdata](#) > save.m

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 1.1 2007/08/29 18:44:38 hewitson Exp \$

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

```
>> pl = save(fsdata, 'Params')
```

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE an fsdata object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for fsdata objects.
0008 %
0009 % CALL:          Save an fsdata object as a .mat file.
0010 %                >> save(d, 'blah.mat')
0011 %                Save an fsdata object as an XML file.
0012 %                >> save(d, 'blah.xml')
0013 %                Save an fsdata object to file specified in plist.
0014 %                >> save(d, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:      $Id: save.m,v 1.1 2007/08/29 18:44:38 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(fsdata, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'fsdata') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041 ALGONAME = mfilename;
0042 VERSION  = '$Id: save.m,v 1.1 2007/08/29 18:44:38 hewitson Exp $';

```



```
0043
0044 if nargin ~= 2
0045     help mfilename;
0046     error('### incorrect inputs');
0047 end
0048
0049 % get object
0050 obj = varargin{1};
0051 if isa(varargin{2}, 'plist')
0052     pl = varargin{2};
0053 elseif ischar(varargin{2})
0054     pl = plist(param('filename', varargin{2}));
0055 else
0056     help mfilename;
0057     error('### incorrect inputs');
0058 end
0059
0060 % Save object
0061 ltpda_saveobj(obj, pl);
0062
0063 % Get default params
0064 function plo = getDefaultPL()
0065
0066 disp('* creating default plist...');
0067 plo = plist();
0068 plo = append(plo, param('filename', 'object.xml'));
0069 disp('* done. ');
0070
0071
0072 % END
```

[Home](#) > [classes](#) > [@timeformat](#) > xml.m

xml

PURPOSE

XML convert a timeformat object to an DOM XML object.

SYNOPSIS

function x = xml(tf)

DESCRIPTION

XML convert a timeformat object to an DOM XML object.

%%%

DESCRIPTION: XML convert a timeformat object to an DOM XML object.

CALL: >> x = xml(tf)

VERSION: \$Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add a time object to the given node of the input XML DOM

This function is called by:

SOURCE CODE

```
0001 function x = xml(tf)
0002 % XML convert a timeformat object to an DOM XML object.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
```

```

0006 % DESCRIPTION: XML convert a timeformat object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(tf)
0009 %
0010 % VERSION: $Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('timeformat');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(tf, x, 'Object', x.docRootNode);
0025
0026
0027 % END

```

[Home](#) > [m](#) > [mysql](#) > ltpda_insert.m

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 1.1 2007/08/31 17:29:22 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [ltpda_obj_submit](#) LTPDA_OBJ_SUBMIT submits the given collection of objects to an LTPDA Repository.

SOURCE CODE

```
0001 function message = ltpda_insert(conn, table, varargin)
0002
0003 % LTPDA_INSERT inserts values into a single row of a table using JDBC driver
0004 % specified by the input connection.
0005 %
0006 % Usage: message = ltpda_insert(conn, table, 'field1', value1, 'field2',
0007 %                               value2,...)
0008 %
```

```
0009 % M Hewitson
0010 %
0011 % 31-07-07
0012 %
0013 % $Id: ltpda_insert.m,v 1.1 2007/08/31 17:29:22 hewitson Exp $
0014 %
0015
0016 q = sprintf('INSERT INTO %s(', table);
0017 for j=1:2:length(varargin)
0018     q = [q varargin{j} ','];
0019 end
0020 q = q(1:end-1);
0021
0022 q = [q ') VALUES('];
0023
0024 for j=2:2:length(varargin)
0025     val = varargin{j};
0026     if ischar(val)
0027         q = [q ' ' val ' ' ','];
0028     elseif isnumeric(val)
0029         q = [q mat2str(val) ','];
0030     else
0031         error('unknown data type');
0032     end
0033
0034 end
0035
0036 q = q(1:end-1);
0037 q = [q ');'];
0038
0039 curs = exec(conn, q);
0040 message = curs.Message;
0041 disp(message);
0042 close(curs);
```

[Home](#) > [classes](#) > [@history](#) > save.m

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 1.1 2007/08/29 18:44:11 hewitson Exp \$

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

```
>> pl = save(history, 'Params')
```

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE a history object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for history objects.
0008 %
0009 % CALL:          Save a history object as a .mat file.
0010 %                >> save(h, 'blah.mat')
0011 %                Save a history object as an XML file.
0012 %                >> save(h, 'blah.xml')
0013 %                Save a history object to file specified in plist.
0014 %                >> save(h, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:      $Id: save.m,v 1.1 2007/08/29 18:44:11 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(history, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'history') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041 ALGONAME = mfilename;
0042 VERSION  = '$Id: save.m,v 1.1 2007/08/29 18:44:11 hewitson Exp $';

```

```
0043
0044 if nargin ~= 2
0045     help mfilename;
0046     error('### incorrect inputs');
0047 end
0048
0049 % get object
0050 obj = varargin{1};
0051 if isa(varargin{2}, 'plist')
0052     pl = varargin{2};
0053 elseif ischar(varargin{2})
0054     pl = plist(param('filename', varargin{2}));
0055 else
0056     help mfilename;
0057     error('### incorrect inputs');
0058 end
0059
0060 % Save object
0061 ltpda_saveobj(obj, pl);
0062
0063 % Get default params
0064 function plo = getDefaultPL()
0065
0066 disp('* creating default plist...');
0067 plo = plist();
0068 plo = append(plo, param('filename', 'object.xml'));
0069 disp('* done. ');
0070
0071
0072 % END
```


[Home](#) > [classes](#) > [@timespan](#) > xml.m

xml

PURPOSE

XML convert a time object to an DOM XML object.

SYNOPSIS

function x = xml(tf)

DESCRIPTION

XML convert a time object to an DOM XML object.

%%%

DESCRIPTION: XML convert a time object to an DOM XML object.

CALL: >> x = xml(tf)

VERSION: \$Id: xml.m,v 1.3 2007/09/15 04:50:18 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add a timespan object to the given node of the input XML DOM

This function is called by:

- [xmladd](#) XMLADD add a timespan object to the given node of the input XML DOM

SOURCE CODE

```
0001 function x = xml(tf)
0002 % XML convert a time object to an DOM XML object.
0003 %
```

```
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert a time object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(tf)
0009 %
0010 % VERSION: $Id: xml.m,v 1.3 2007/09/15 04:50:18 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('timespan');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(tf, x, 'Object', x.docRootNode);
0025
0026
0027 % END
```

[Home](#) > [classes](#) > [@pole](#) > display.m

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 1.4 2008/01/02 17:58:51 ingo 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:

SOURCE CODE

```
0001 function txt = display(p, varargin)
0002 % DISPLAY overloads display functionality for pole objects.
```

```

0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY overloads display functionality for pole objects.
0007 %
0008 % CALL:          txt = display(pole)
0009 %
0010 % INPUT:         pole - pole object
0011 %
0012 % OUTPUT:        txt - cell array with strings to display the pole object
0013 %
0014 % VERSION:       $Id: display.m,v 1.4 2008/01/02 17:58:51 ingo Exp $
0015 %
0016 % HISTORY:       30-01-2007 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: display.m,v 1.4 2008/01/02 17:58:51 ingo Exp $';
0022
0023 % Check if this is a call for parameters
0024 if nargin == 2
0025     if isa(p, 'pole') && ischar(varargin{1})
0026         in = char(varargin{1});
0027         if strcmp(in, 'Params')
0028             txt = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             txt = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 txt = {};
0038
0039 for i=1:numel(p)
0040     banner_start = sprintf('---- pole %d ----', i);
0041     txt{end+1} = banner_start;
0042
0043     % get key and value
0044     name = get(p(i), 'name');
0045     f     = get(p(i), 'f');
0046     q     = get(p(i), 'q');
0047     ri    = get(p(i), 'ri');
0048
0049     % display
0050     if ~isempty(f)
0051         if q > 0.5
0052             txt{end+1} = [ sprintf('%s: %g Hz, Q=%g [' , name, f, q) num2str(ri(1))
0053                 ']]];
0054         else

```

```
0054     txt{end+1} = sprintf('%s: %g Hz', name, f);
0055     end
0056 end
0057
0058     banner_end(1:length(banner_start)) = '-';
0059     txt{end+1} = banner_end;
0060
0061 end
0062
0063 if nargout == 0
0064     for ii=1:length(txt)
0065         disp(txt{ii});
0066     end
0067 end
0068
```

[Home](#) > [m](#) > [helper](#) > ltpda_io_collect.m

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.

%%%

LTPDA_IO_COLLECT collects all AOs and plists from the inputs and returns
a cell array of AOs and a single combined plist.

```
>> [aos, pl] = ltpda_io_collect(args)
```

Inputs:

args - cell array containing matrices or vectors of AOs or plists

Outputs:

aos - a cell array of analysis objects, one for each input

pl - a single plist which combines all the input plists

M Hewitson 09-10-07

\$Id: ltpda_io_collect.m,v 1.1 2007/10/08 09:35:48 hewitson Exp \$

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SUBFUNCTIONS

- [function aos = unpack_aos\(ain\)](#)

SOURCE CODE

```

0001 function varargout = ltpda_io_collect(varargin)
0002
0003 % LTPDA_IO_COLLECT collects all AOs and plists from the inputs.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % LTPDA_IO_COLLECT collects all AOs and plists from the inputs and returns
0007 %          a cell array of AOs and a single combined plist.
0008 %
0009 %   >> [aos, pl] = ltpda_io_collect(args)
0010 %
0011 %   Inputs:
0012 %       args - cell array containing matrices or vectors of AOs or plists
0013 %
0014 %   Outputs:
0015 %       aos  - a cell array of analysis objects, one for each input
0016 %       pl   - a single plist which combines all the input plists
0017 %
0018 %
0019 % M Hewitson 09-10-07
0020 %
0021 % $Id: ltpda_io_collect.m,v 1.1 2007/10/08 09:35:48 hewitson Exp $
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 aos = {};
0026 pl  = [];
0027
0028 % Go through inputs
0029 for ii=1:nargin
0030     a = varargin{ii};
0031     if isa(a, 'ao')
0032         aos = [aos unpack_aos(a)];
0033     elseif isa(a, 'plist')
0034         pl = combine(pl, a);
0035     else
0036         disp(a);
0037         warning(['! unrecognised input ' num2str(ii) ': ']);
0038     end
0039
0040 end
0041
0042 % Set outputs
0043 if nargout == 2
0044     varargout{1} = aos;
0045     varargout{2} = pl;
0046 else
0047     help(mfilename);
0048     error('### incorrect outputs');
0049 end

```

```
0050
0051
0052 %-----
0053 % Unpack AOs from input
0054 function aos = unpack_aos(ain)
0055
0056 aos = {};
0057 for jj=1:numel(ain)
0058     a = ain(jj);
0059     if numel(a) > 1
0060         aos = [aos unpack_aos(a)];
0061     else
0062         aos = [aos {a}];
0063     end
0064 end
0065
```


[Home](#) > [classes](#) > [@mfir](#) > save.m

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 1.2 2007/12/10 18:07:17 ingo Exp \$

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

```
>> pl = save(mfir, 'Params')
```

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

```

0001 function varargout = save(varargin)
0002 % SAVE an mfir object to file.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SAVE overloads the save method for mfir objects.
0007 %
0008 % CALL:          Save a mfir object as a .mat file.
0009 %                >> save(f, 'blah.mat')
0010 %                Save a mfir object as an XML file.
0011 %                >> save(f, 'blah.xml')
0012 %                Save a mfir object to file specified in plist.
0013 %                >> save(f, plist)
0014 %
0015 % Parameters: 'filename' - name of the file
0016 %
0017 % VERSION:      $Id: save.m,v 1.2 2007/12/10 18:07:17 ingo Exp $
0018 %
0019 % The following call returns a parameter list object that contains the
0020 % default parameter values:
0021 %
0022 % >> pl = save(mfir, 'Params')
0023 %
0024 % HISTORY: 28-08-07 M Hewitson
0025 %          Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 ALGONAME = mfilename;
0030 VERSION  = '$Id: save.m,v 1.2 2007/12/10 18:07:17 ingo Exp $';
0031
0032 % Check if this is a call for parameters
0033 if nargin == 2
0034     if isa(varargin{1}, 'mfir') && ischar(varargin{2})
0035         in = char(varargin{2});
0036         if strcmp(in, 'Params')
0037             varargout{1} = getDefaultPL();
0038             return
0039         end
0040     end
0041 end
0042

```

```
0043 if nargin ~= 2
0044     help mfilename;
0045     error('### incorrect inputs');
0046 end
0047
0048 % get object
0049 obj = varargin{1};
0050 if isa(varargin{2}, 'plist')
0051     pl = varargin{2};
0052 elseif ischar(varargin{2})
0053     pl = plist(param('filename', varargin{2}));
0054 else
0055     help mfilename;
0056     error('### incorrect inputs');
0057 end
0058
0059 % Save object
0060 ltpda_saveobj(obj, pl);
0061
0062 % Get default params
0063 function plo = getDefaultPL()
0064
0065 disp('* creating default plist...');
0066 plo = plist();
0067 plo = append(plo, param('filename', 'object.xml'));
0068 disp('* done. ');
0069
0070
0071 % END
```

[Home](#) > [classes](#) > [@tsdata](#) > xml.m

xml

PURPOSE

XML convert a tsdata object to an DOM XML object.

SYNOPSIS

function x = xml(d)

DESCRIPTION

XML convert a tsdata object to an DOM XML object.

%%%

DESCRIPTION: XML convert a tsdata object to an DOM XML object.

CALL: >> x = xml(d)

VERSION: \$Id: xml.m,v 1.2 2007/09/15 04:50:19 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add a tsdata object to the given node of the input XML DOM

This function is called by:

- [xmladd](#) XMLADD add a tsdata object to the given node of the input XML DOM

SOURCE CODE

```
0001 function x = xml(d)
0002 % XML convert a tsdata object to an DOM XML object.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert a tsdata object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(d)
0009 %
0010 % VERSION: $Id: xml.m,v 1.2 2007/09/15 04:50:19 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('tsdata');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(d, x, 'Object', x.docRootNode);
0025
0026
0027 % END

```

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 1.8 2008/01/08 16:39:11 ingo 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:

SOURCE CODE

```
0001 function txt = display(ps, varargin)
0002 % DISPLAY overload terminal display for provenance objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY overload terminal display for provenance objects.
0007 %
0008 % VERSION: $Id: display.m,v 1.8 2008/01/08 16:39:11 ingo Exp $
```

```

0009 %
0010 % HISTORY: 07-05-2007 M Hewitson
0011 %           Creation
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014
0015 VERSION = '$Id: display.m,v 1.8 2008/01/08 16:39:11 ingo Exp $';
0016
0017 % Check if this is a call for parameters
0018 if nargin == 2
0019     if isa(ps, 'provenance') && ischar(varargin{1})
0020         in = char(varargin{1});
0021         if strcmp(in, 'Params')
0022             txt = plist;
0023             return
0024         elseif strcmp(in, 'Version')
0025             txt = VERSION;
0026             return
0027         end
0028     end
0029 end
0030
0031 txt = {};
0032
0033 for i=1:numel(ps)
0034     p = ps(i);
0035     banner = sprintf('----- provenance %02d -----', i);
0036     txt{end+1} = banner;
0037
0038     txt{end+1} = ' ';
0039
0040     txt{end+1} = sprintf('      creator:  %s', get(p, 'creator'));
0041     txt{end+1} = sprintf('      created:  %s', char(get(p, 'created')));
0042     txt{end+1} = sprintf('      ip:      %s', get(p, 'ip'));
0043     txt{end+1} = sprintf('      hostname: %s', get(p, 'hostname'));
0044     txt{end+1} = sprintf('      os:      %s', get(p, 'os'));
0045     txt{end+1} = sprintf('      MATLAB ver: %s', get(p, 'matlab_version'));
0046     txt{end+1} = sprintf('      Sig Proc ver: %s', get(p, 'sigproc_version'));
0047     txt{end+1} = sprintf('      LTPDA ver:  %s', get(p, 'ltpda_version'));
0048
0049     banner_end(1:length(banner)) = '-';
0050     txt{end+1} = banner_end;
0051
0052     txt{end+1} = ' ';
0053     txt{end+1} = ' ';
0054 end
0055
0056 if nargout == 0
0057     for ii=1:length(txt)
0058         disp(txt{ii});
0059     end

```

```
0060 end
0061
0062 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [helper](#) > ltpda_isobject.m

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 1.2 2007/09/15 04:51:20 hewitson Exp \$

HISTORY: 09-05-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SUBFUNCTIONS

- `function varargout = isltpdaobj(obj)`

SOURCE CODE

```

0001 function varargout = ltpda_isobject(varargin)
0002
0003 % LTPDA_ISOBJECT checks that the input objects are one of the LTPDA object
0004 % types.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: LTPDA_ISOBJECT checks that the input objects are one of the
0009 % LTPDA object
0010 % types.
0011 % CALL:      result = ltpda_isobject(a1)
0012 %           classes = ltpda_isobject()
0013 %
0014 % INPUTS:    objects
0015 %
0016 % OUTPUTS:   result == 1 if all input objects are LTPDA objects
0017 %           result == 0 otherwise
0018 %           classes - a list of recognised LTPDA object types
0019 %
0020 % VERSION:   $Id: ltpda_isobject.m,v 1.2 2007/09/15 04:51:20 hewitson Exp $
0021 %
0022 % HISTORY:   09-05-07 M Hewitson
0023 %           Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026
0027
0028 if nargin == 0
0029     varargout{1} = classes;
0030 else
0031
0032     for j=1:nargin
0033         % check this object
0034         obj = varargin{1};
0035         if iscell(obj)
0036             for k=1:length(obj)
0037                 % check each one
0038                 if ~isltpdaobj(obj{k})
0039                     varargout{1} = 0;
0040                     return;
0041                 end
0042             end
0043         else
0044             for k=1:length(obj)
0045                 % check each one

```

```

0046
0047     % Stupid hack required because database objects can't cope with
0048     % begin indexed like conn(1) - all other objects work for this
0049     % call.
0050     if k==1
0051         if ~isltpdaobj(obj)
0052             varargout{1} = 0;
0053             return
0054         end
0055     else
0056         if ~isltpdaobj(obj(k))
0057             varargout{1} = 0;
0058             return;
0059         end
0060     end
0061 end
0062 end
0063 end
0064 end
0065
0066 varargout{1} = 1;
0067
0068 %-----
0069 % Test a single object
0070 %
0071 function varargout = isltpdaobj(obj)
0072
0073 classes = {'ao',...
0074            'param',...
0075            'plist',...
0076            'cdata',...
0077            'tsdata',...
0078            'fsdata',...
0079            'xydata',...
0080            'specwin',...
0081            'mfir',...
0082            'miir',...
0083            'pzmodel',...
0084            'pole',...
0085            'zero',...
0086            'provenance',...
0087            'time',...
0088            'timeformat',...
0089            'timespan',...
0090            'history'};
0091
0092 found = 0;
0093 for k=1:length(classes)
0094     type = classes{k};
0095     if isa(obj, type)
0096         found = 1;
0097     end

```

```
0098         end
0099         if ~found
0100             varargout{1} = 0;
0101             return
0102         else
0103             varargout{1} = 1;
0104         end
0105
```

[Home](#) > [classes](#) > [@miir](#) > save.m

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 1.1 2007/08/29 18:43:12 hewitson Exp \$

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

```
>> pl = save(miir, 'Params')
```

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE an miir object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for miir objects.
0008 %
0009 % CALL:          Save a miir object as a .mat file.
0010 %                >> save(f, 'blah.mat')
0011 %                Save a miir object as an XML file.
0012 %                >> save(f, 'blah.xml')
0013 %                Save a miir object to file specified in plist.
0014 %                >> save(f, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:      $Id: save.m,v 1.1 2007/08/29 18:43:12 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(miir, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'miir') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041 ALGONAME = mfilename;
0042 VERSION  = '$Id: save.m,v 1.1 2007/08/29 18:43:12 hewitson Exp $';

```

```
0043
0044 if nargin ~= 2
0045     help mfilename;
0046     error('### incorrect inputs');
0047 end
0048
0049 % get object
0050 obj = varargin{1};
0051 if isa(varargin{2}, 'plist')
0052     pl = varargin{2};
0053 elseif ischar(varargin{2})
0054     pl = plist(param('filename', varargin{2}));
0055 else
0056     help mfilename;
0057     error('### incorrect inputs');
0058 end
0059
0060 % Save object
0061 ltpda_saveobj(obj, pl);
0062
0063 % Get default params
0064 function plo = getDefaultPL()
0065
0066 disp('* creating default plist...');
0067 plo = plist();
0068 plo = append(plo, param('filename', 'object.xml'));
0069 disp('* done. ');
0070
0071
0072 % END
```

[Home](#) > [classes](#) > [@xydata](#) > xml.m

xml

PURPOSE

XML convert an xydata object to an DOM XML object.

SYNOPSIS

function x = xml(d)

DESCRIPTION

XML convert an xydata object to an DOM XML object.

%%%

DESCRIPTION: XML convert an xydata object to an DOM XML object.

CALL: >> x = xml(d)

VERSION: \$Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add a xydata object to the given node of the input XML DOM

This function is called by:

- [xmladd](#) XMLADD add a xydata object to the given node of the input XML DOM

SOURCE CODE

```
0001 function x = xml(d)
0002 % XML convert an xydata object to an DOM XML object.
0003 %
```



```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert an xydata object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(d)
0009 %
0010 % VERSION: $Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('xydata');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(d, x, 'Object', x.docRootNode);
0025
0026
0027 % END

```

[Home](#) > [classes](#) > [@pzmodel](#) > display.m

display

PURPOSE

DISPLAY overloads display functionality for pzmodel objects.

SYNOPSIS

function txt = display(p, varargin)

DESCRIPTION

```
DISPLAY overloads display functionality for pzmodel objects.

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

DESCRIPTION: DISPLAY overloads display functionality for pzmodel objects.

CALL:      txt      = display(pzmodel)

INPUT:     pzmodel - pole/zero model object

OUTPUT:    txt      - cell array with strings to display the pole/zero model object

VERSION:   $Id: display.m,v 1.5 2008/01/07 18:13:10 ingo Exp $

HISTORY:   30-01-2007 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:

SOURCE CODE

```
0001 function txt = display(p, varargin)
0002 % DISPLAY overloads display functionality for pzmodel objects.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY overloads display functionality for pzmodel objects.
0007 %
0008 % CALL:          txt      = display(pzmodel)
0009 %
0010 % INPUT:         pzmodel - pole/zero model object
0011 %
0012 % OUTPUT:        txt      - cell array with strings to display the pole/zero model
object
0013 %
0014 % VERSION:       $Id: display.m,v 1.5 2008/01/07 18:13:10 ingo Exp $
0015 %
0016 % HISTORY:       30-01-2007 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: display.m,v 1.5 2008/01/07 18:13:10 ingo Exp $';
0022
0023 % Check if this is a call for parameters
0024 if nargin == 2
0025     if isa(p, 'pzmodel') && ischar(varargin{1})
0026         in = char(varargin{1});
0027         if strcmp(in, 'Params')
0028             txt = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             txt = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 txt = {};
0038
0039 for i=1:numel(p)
0040     banner = sprintf('---- pzmodel %d ----', i);
0041     txt{end+1} = banner;
0042
0043     % get key and value
0044     name = get(p(i), 'name');
0045     g     = get(p(i), 'gain');
0046     ps    = get(p(i), 'poles');
0047     zs    = get(p(i), 'zeros');
0048     np    = length(ps);
0049     nz    = length(zs);
0050
0051     % display
0052     txt{end+1} = ['model:      ' name];
0053     txt{end+1} = ['gain :      ' num2str(g)];
0054     for j=1:np

```

```
0055     txt{end+1} = [sprintf('pole %03d: ', j) char(ps(j)) ];
0056 end
0057 for j=1:nz
0058     txt{end+1} = [sprintf('zero %03d: ', j) char(zs(j)) ];
0059 end
0060
0061 banner_end(1:length(banner)) = '-';
0062 txt{end+1} = banner_end;
0063
0064 end
0065
0066 if nargout == 0
0067     for ii=1:length(txt)
0068         disp(txt{ii});
0069     end
0070 end
0071
```

[Home](#) > [m](#) > [helper](#) > ltpda_isuserobject.m

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 1.1 2007/11/19 09:49:47 hewitson Exp \$

HISTORY: 09-05-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SUBFUNCTIONS

- function varargout = isltpdaobj(obj)

SOURCE CODE

```

0001 function varargout = ltpda_isuserobject(varargin)
0002
0003 % LTPDA_ISUSEROBJECT checks that the input objects are one of the LTPDA user
object
0004 % types.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: LTPDA_ISUSEROBJECT checks that the input objects are one of the
0009 %               LTPDA user object types.
0010 %
0011 % CALL:        result = ltpda_isuserobject(a1)
0012 %               classes = ltpda_isuserobject()
0013 %
0014 % INPUTS:      objects
0015 %
0016 % OUTPUTS:     result == 1 if all input objects are LTPDA objects
0017 %               result == 0 otherwise
0018 %               classes - a list of recognised LTPDA object types
0019 %
0020 % VERSION:     $Id: ltpda_isuserobject.m,v 1.1 2007/11/19 09:49:47 hewitson Exp
$
0021 %
0022 % HISTORY: 09-05-07 M Hewitson
0023 %           Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026
0027
0028 if nargin == 0
0029     varargout{1} = classes;
0030 else
0031
0032     for j=1:nargin
0033         % check this object
0034         obj = varargin{1};
0035         if iscell(obj)
0036             for k=1:length(obj)
0037                 % check each one
0038                 if ~isltpdaobj(obj{k})
0039                     varargout{1} = 0;
0040                     return;
0041                 end
0042             end
0043         else
0044             for k=1:length(obj)
0045                 % check each one
0046

```

```

0047      % Stupid hack required because database objects can't cope with
0048      % begin indexed like conn(1) - all other objects work for this
0049      % call.
0050      if k==1
0051          if ~isltpdaobj(obj)
0052              varargout{1} = 0;
0053              return
0054          end
0055      else
0056          if ~isltpdaobj(obj(k))
0057              varargout{1} = 0;
0058              return;
0059          end
0060      end
0061  end
0062  end
0063  end
0064  end
0065
0066  varargout{1} = 1;
0067
0068  %-----
0069  % Test a single object
0070  %
0071  function varargout = isltpdaobj(obj)
0072
0073  classes = {'ao',...
0074             'plist',...
0075             'specwin',...
0076             'mfir',...
0077             'miir',...
0078             'pzmodel',...
0079             'time',...
0080             'timespan'};
0081
0082      found = 0;
0083      for k=1:length(classes)
0084          type = classes{k};
0085          if isa(obj, type)
0086              found = 1;
0087          end
0088      end
0089      if ~found
0090          varargout{1} = 0;
0091          return
0092      else
0093          varargout{1} = 1;
0094      end
0095

```

[Home](#) > [classes](#) > [@zero](#) > xml.m

xml

PURPOSE

XML convert a zero object to an DOM XML object.

SYNOPSIS

function x = xml(z)

DESCRIPTION

XML convert a zero object to an DOM XML object.

%%%

DESCRIPTION: XML convert a zero object to an DOM XML object.

CALL: >> x = xml(z)

VERSION: \$Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
Creation

SEE also xmlwrite, xmlread

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmladd](#) XMLADD add a zero object to the given node of the input XML DOM

This function is called by:

- [xmladd](#) XMLADD add a zero object to the given node of the input XML DOM

SOURCE CODE

```
0001 function x = xml(z)
0002 % XML convert a zero object to an DOM XML object.
0003 %
```



```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XML convert a zero object to an DOM XML object.
0007 %
0008 % CALL: >> x = xml(z)
0009 %
0010 % VERSION: $Id: xml.m,v 1.2 2007/09/15 04:50:18 hewitson Exp $
0011 %
0012 % HISTORY: 28-08-2007 M Hewitson
0013 %           Creation
0014 %
0015 % SEE also xmlwrite, xmlread
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 %% Convert to XML
0020 x.docNode = com.mathworks.xml.XMLUtils.createDocument('zero');
0021 x.docRootNode = x.docNode.getDocumentElement;
0022 % x.docNode.appendChild(x.docNode.createComment(sprintf('Created %s', datestr
(now))));
0023
0024 x = xmladd(z, x, 'Object', x.docRootNode);
0025
0026
0027 % END

```

[Home](#) > [classes](#) > [@specwin](#) > display.m

display

PURPOSE

DISPLAY overloads display functionality for specwin objects.

SYNOPSIS

function txt = display(ws, varargin)

DESCRIPTION

DISPLAY overloads display functionality for specwin objects.

%%%

DESCRIPTION: DISPLAY overloads display functionality for specwin objects.

CALL: txt = display(specwin)

INPUT: specwin - spectral window object

OUTPUT: txt - cell array with strings to display the spectral window object

VERSION: \$Id: display.m,v 1.6 2008/01/08 16:40:29 ingo Exp \$

HISTORY: 30-01-2007 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a specwin object into a string.
- [get](#) GET get specwin object properties.

This function is called by:

SOURCE CODE

```
0001 function txt = display(ws, varargin)
0002 % DISPLAY overloads display functionality for specwin objects.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY overloads display functionality for specwin objects.
0007 %
0008 % CALL:          txt      = display(specwin)
0009 %
0010 % INPUT:         specwin - spectral window object
0011 %
0012 % OUTPUT:        txt      - cell array with strings to display the spectral window
object
0013 %
0014 % VERSION:       $Id: display.m,v 1.6 2008/01/08 16:40:29 ingo Exp $
0015 %
0016 % HISTORY:       30-01-2007 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: display.m,v 1.6 2008/01/08 16:40:29 ingo Exp $';
0022
0023 % Check if this is a call for parameters
0024 if nargin == 2
0025     if isa(ws, 'specwin') && ischar(varargin{1})
0026         in = char(varargin{1});
0027         if strcmp(in, 'Params')
0028             txt = plist;
0029             return
0030         elseif strcmp(in, 'Version')
0031             txt = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 txt = {};
0038
0039 for i=1:numel(ws)
0040     w = ws(i);
0041     banner = sprintf('----- %s -----', get(w, 'name'));
0042     txt{end+1} = banner;
0043
0044     txt{end+1} = ' ';
0045     txt{end+1} = sprintf('    alpha: %g', w.alpha);
0046     txt{end+1} = sprintf('    psll: %g', w.psll);
0047     txt{end+1} = sprintf('    rov: %g', w.rov);
0048     txt{end+1} = sprintf('    nenbw: %g', w.nenbw);
0049     txt{end+1} = sprintf('    w3db: %g', w.w3db);
0050     txt{end+1} = sprintf('flatness: %g', w.flatness);
0051     txt{end+1} = sprintf('    ws: %g', w.ws);
0052     txt{end+1} = sprintf('    ws2: %g', w.ws2);
0053     txt{end+1} = sprintf('    win: %d', length(w.win));
0054
0055     txt{end+1} = ' ';

```

```
0056
0057     banner_end(1:length(banner)) = '-';
0058     txt{end+1} = banner_end;
0059
0060     txt{end+1} = ' ';
0061     txt{end+1} = ' ';
0062 end
0063
0064 if nargout == 0
0065     for ii=1:length(txt)
0066         disp(txt{ii});
0067     end
0068 end
0069
0070 % END
```

[Home](#) > [m](#) > [gui](#) > [ao_browser](#) > ltpda_javacomponent.m

ltpda_javacomponent

PURPOSE

WARNING: This feature is not supported in MATLAB

SYNOPSIS

function [hcomponent, hcontainer] = ltpda_javacomponent(component, position, parent)

DESCRIPTION

WARNING: This feature is not supported in MATLAB and the API and functionality may change in a future release.

JAVACOMPONENT Create a Java AWT Component and put it in a figure

JAVACOMPONENT(COMPONENTNAME) creates the Java component specified by the string COMPONENTNAME and places it in the current figure or creates a new figure if one is not available. The default position is [20 20 60 20].

NOTE: This is a thread safe way to create and embed a java component in a figure window. For more thread safe functions to create and modify java components, see AWTCREATE, AWTINVOKE.

JAVACOMPONENT(HCOMPONENT) places the Java component HCOMPONENT in the current figure or creates a new figure if one is not available. The default position is [20 20 60 20].

JAVACOMPONENT(..., POSITION, PARENT) places the Java component in the specified PARENT at position POSITION. PARENT can be a Figure or a Uipanel. POSITION is in pixel units with the format [left, bottom, width, height].

JAVACOMPONENT(..., CONSTRAINT, PARENT) places the Java component next to the figure's drawing area using CONSTRAINT. CONSTRAINT that can be NORTH, SOUTH, EAST, OR WEST placement - following Java AWT's BorderLayout rules. The handle to the Java component is returned on success, empty is returned on error. If the parent is a uipanel, the component is placed in the parent figure of the uipanel. If parent is a uitoolbar handle, CONSTRAINT is ignored and the component is placed last in the child list for the given toolbar.

[HCOMPONENT, HCONTAINER] = JAVACOMPONENT(...) returns the handle to the Java component in HCOMPONENT and its HG container in HCONTAINER. HCONTAINER is only returned when pixel positioning is used. It should be used to change the units, position, and visibility of the Java component after it is added to the figure.

Examples:

```
f = figure;
b = javacomponent('javax.swing.JButton'); % Thread safe creation
set(b,'ActionPerformedCallback','disp Hi!');

f = figure('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: 1.1 \$ \$Date: 2007/06/14 14:53:32 \$

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, parentsIsFigure)
- function str=usage

SOURCE CODE

```

0001 function [hcomponent, hcontainer] = ltpda_javacomponent(component, position,
parent)
0002 % WARNING: This feature is not supported in MATLAB
0003 % and the API and functionality may change in a future release.
0004 %
0005 % JAVACOMPONENT Create a Java AWT Component and put it in a figure
0006 %
0007 % JAVACOMPONENT(COMPONENTNAME) creates the Java component specified by the
0008 % string COMPONENTNAME and places it in the current figure or creates a new
0009 % figure if one is not available. The default position is [20 20 60 20].
0010 % NOTE: This is a thread safe way to create and embed a java component in a
0011 % figure window. For more thread safe functions to create and modify java
0012 % components, see AWTCREATE, AWTINVOKE.
0013 %
0014 % JAVACOMPONENT(HCOMPONENT) places the Java component HCOMPONENT in the
0015 % current figure or creates a new figure if one is not available. The
0016 % default position is [20 20 60 20].
0017 %
0018 % JAVACOMPONENT(..., POSITION, PARENT) places the Java component in the
0019 % specified PARENT at position POSITION. PARENT can be a Figure or a
0020 % Uipanel. POSITION is in pixel units with the format [left, bottom, width,
0021 % height].
0022 %
0023 % JAVACOMPONENT(..., CONSTRAINT, PARENT) places the Java component next to
0024 % the figure's drawing area using CONSTRAINT. CONSTRAINT that can be NORTH,
0025 % SOUTH, EAST, OR WEST placement - following Java AWT's BorderLayout rules.
0026 % The handle to the Java component is returned on success, empty is
0027 % returned on error. If the parent is a uipanel, the component is placed in
0028 % the parent figure of the uipanel. If parent is a uitoolbar handle,
0029 % CONSTRAINT is ignored and the component is placed last in the child list
0030 % for the given toolbar.
0031 %
0032 % [HCOMPONENT, HCONTAINER] = JAVACOMPONENT(...)
0033 % returns the handle to the Java component in HCOMPONENT and its HG
0034 % container in HCONTAINER. HCONTAINER is only returned when pixel
0035 % positioning is used. It should be used to change the units, position,
0036 % and visibility of the Java component after it is added to the figure.
0037 %
0038 %
0039 %   Examples:
0040 %
0041 %   f = figure;

```

```

0042 %     b = javacomponent('javax.swing.JButton'); % Thread safe creation
0043 %     set(b,'ActionPerformedCallback','disp Hi!');
0044 %
0045 %     f = figure('WindowStyle', 'docked');
0046 %     b1 = awtcreate('javax.swing.JButton', 'Ljava.lang.String;', 'Hi!');
0047 %     set(b1,'ActionPerformedCallback','disp Hi!');
0048 %     javacomponent(b1);
0049 %
0050 %     f = figure;
0051 %     comp = awtcreate('javax.swing.JSpinner');
0052 %     [comp, container] = javacomponent(comp);
0053 %     set(container,'Position', [100, 100, 100, 40]);
0054 %     set(container,'Units', 'normalized');
0055 %
0056 %     f = figure;
0057 %     p = uipanel('Position', [0 0 .2 1]);
0058 %     ppos = getpixelposition(p);
0059 %     tr = awtcreate('javax.swing.JTree');
0060 %     [tree treecontainer] = javacomponent(tr, [0 0 ppos(3) ppos(4)], p);
0061 %
0062 %     f = figure('WindowStyle', 'docked');
0063 %     ta = awtcreate('javax.swing.JTable', 'II', 3, 10);
0064 %     table = javacomponent(ta, java.awt.BorderLayout.SOUTH, f);
0065 %
0066 %     f = figure;
0067 %     tb = uitoolbar(f);
0068 %     b2 = awtcreate('javax.swing.JButton', 'Ljava.lang.String;', 'Hi again!');
0069 %     b2 = javacomponent(b2, [], tb); % Note: Position is ignored.
0070 %     set(b2,'ActionPerformedCallback','disp(''Hi again!'')');
0071 %
0072 % See also USEJAVACOMPONENT, AWTCREATE, AWTINVOKE
0073 %
0074 % Copyright 1984-2006 The MathWorks, Inc.
0075 % $Revision: 1.1 $ $Date: 2007/06/14 14:53:32 $
0076
0077 if (usejavacomponent == 0)
0078     err.message = 'JAVACOMPONENT is not supported on this platform';
0079     err.identifier = 'MATLAB:javacomponent:FeatureNotSupported';
0080     error(err);
0081 end
0082
0083 if ~isempty(nargchk(1,3,nargin))
0084     error('MATLAB:javacomponent',usage);
0085 end
0086
0087 if nargin < 3
0088     parent = gcf;
0089 end
0090
0091 if nargin < 2
0092     position = [20 20 60 20];
0093 end
0094

```



```

0095 parentIsFigure = false;
0096 hParent = handle(parent);
0097 if ( isa(hParent, 'figure') || ...
0098     isa(hParent, 'uicontainer') || ...
0099     isa(hParent, 'uiflowcontainer') || ...
0100     isa(hParent, 'uigridcontainer'))
0101     parentIsFigure = true;
0102     peer = get(ancestor(parent, 'figure'), 'JavaFrame');
0103 elseif isa(hParent, 'uitoolbar')
0104     peer = get(parent, 'JavaContainer');
0105     if isempty(peer)
0106         drawnow;
0107         peer = get(parent, 'JavaContainer');
0108     end
0109 else
0110     error('MATLAB:javacomponent:InvalidParentHandle', 'Invalid parent handle\n%s', usage)
0111 end
0112
0113 if isempty(peer)
0114     error('MATLAB:javacomponent:JavaFigsNotEnabled', 'Java figures are not enabled')
0115 end
0116
0117 hUicontainer = [];
0118 hgp = [];
0119 returnContainer = 1;
0120
0121 if ischar(component)
0122     component = awtcreate(component);
0123 end
0124
0125 % promote the component to a handle object first.
0126 % It seems once a java object is cast to a handle, you cannot get another
0127 % handle with 'callbackproperties'. So, do want you need outright and use
0128 % the same handle in the subsequent code.
0129 if ~isjava(component)
0130     component = java(component);
0131 end
0132 hcomponent = handle(component, 'callbackProperties');
0133
0134 if nargin == 1
0135     hgp = handle(peer.addchild(component));
0136     % parent must be a figure, we default to(gcf upstairs
0137     createPanel;
0138     hgp.setUicontainer(hUicontainer);
0139 else
0140     if parentIsFigure
0141         if isnumeric(position)
0142             if isempty(position)
0143                 position = [20 20 60 20];
0144             end
0145             % numeric position is not set here, rely on the uicontainer

```

```

0146         % listeners below.
0147         hgp = handle(peer.addchild(component));
0148         createPanel;
0149         hgp.setUIContainer(hUicontainer);
0150     elseif ...
0151         isequal(char(position),char(java.awt.BorderLayout.NORTH)) || ...
0152         isequal(char(position),char(java.awt.BorderLayout.SOUTH)) || ...
0153         isequal(char(position),char(java.awt.BorderLayout.EAST)) || ...
0154         isequal(char(position),char(java.awt.BorderLayout.WEST))
0155         hgp = handle(peer.addchild(component, position));
0156         returnContainer = 0;
0157         createPanel;
0158     else
0159         error('MATLAB:javacomponent:InvalidPosition', 'Invalid component
position\n%s', usage)
0160     end
0161 else
0162     % Adding component to the toolbar.
0163     % component position is ignored for now
0164     peer.add(component);
0165     hUicontainer = parent; % toolbar.
0166     handles = getappdata(hUicontainer, 'childhandles');
0167     handles = [handles, hcomponent];
0168     setappdata(hUicontainer, 'childhandles', handles);
0169 end
0170
0171 % make sure the component is on the screen so the
0172 % caller can interact with it right now.
0173 % drawnow;
0174 end
0175
0176 configureComponent;
0177
0178 if (returnContainer == 1)
0179     hcontainer = hUicontainer;
0180 else
0181     hcontainer = [];
0182 end
0183
0184 function createPanel
0185     % add delete listener
0186     hUicontainer = hgjavacomponent('Parent', parent, 'Units', 'Pixels');
0187
0188     set(hUicontainer, 'UserData', char(component.getClass.getName)); % For
findobj queries.
0189     if isa(java(hgp), 'com.mathworks.hg.peer.FigureChild')
0190         set(hUicontainer, 'FigureChild', hgp);
0191     end
0192     if isa(java(hcomponent), 'javax.swing.JComponent')
0193         % force component to be opaque if it's a JComponent. This prevents
0194         % lightweight component from showing the figure background (which
0195         % may never get a paint event)

```

```

0196         hcomponent.setOpaque(true);
0197     end
0198     set(hUicontainer, 'JavaPeer', hcomponent);
0199
0200     if (returnContainer == 1)
0201         % add resize listener to parent (parent must be a figure or this
dies quietly)
0202         % this is for normalized units
0203         addlistener(hUicontainer, 'PixelBounds', 'PropertyPostSet',
@handleResize);
0204
0205         % add visible listener
0206         addlistener(hUicontainer, 'Visible', 'PropertyPostSet',
@handleVisible);
0207
0208         % add parent listener
0209         addlistener(hUicontainer, 'Parent', 'PropertyPreSet',
@handlePreParent);
0210
0211         % add parent listener
0212         addlistener(hUicontainer, 'Parent', 'PropertyPostSet',
@handlePostParent);
0213
0214         % force though 1st resize event
0215         set(hUicontainer, 'Position', position);
0216     else
0217         % For the BorderLayout components, we dont really want the
0218         % hUicontainer to show. But, it provides a nice place for cleanup.
0219         set(hUicontainer, 'Visible', 'off', 'Handlevisibility', 'off');
0220         % Set position out of the figure to work around a current bug
0221         % due to which invisible uicontainers show up when renderer is
0222         % OpenGL (G.
0223         set(hUicontainer, 'Position', [1 1 0.01 0.01]);
0224     end
0225
0226     if isa(component, 'com.mathworks.hg.peer.FigureChild')
0227         component.setUicontainer(hUicontainer);
0228     end
0229
0230     function handleResize(obj, evd) %#ok - mlint
0231         hgp.setPixelPosition(getpixelposition(hUicontainer, true));
0232     end
0233
0234     function handleVisible(obj, evd) %#ok - mlint
0235         hgp.setVisible(strcmp(get(hUicontainer, 'Visible'), 'on'))
0236     end
0237
0238     function handlePreParent(obj, evd) %#ok - mlint
0239         oldfig = ancestor(parent, 'figure');
0240         newfig = ancestor(evd.NewValue, 'figure');
0241         if ~isempty(newfig) && ~isequal(oldfig, newfig)
0242             peer = get(oldfig, 'JavaFrame');

```

```

0243         peer.remove(component);
0244     end
0245 end
0246
0247 function handlePostParent(obj, evd) %#ok - mlint
0248     oldfig = ancestor(parent, 'figure');
0249     newfig = ancestor(evd.NewValue, 'figure');
0250     if ~isempty(newfig) && ~isequal(oldfig, newfig)
0251         peer = get(newfig, 'JavaFrame');
0252         hgp= handle(peer.addchild(component));
0253         if isa(java(hgp), 'com.mathworks.hg.peer.FigureChild')
0254             % used by the uicontainer C-code
0255             setappdata(hUicontainer, 'FigureChild', java(hgp));
0256         end
0257         parent = newfig;
0258     end
0259     hgp.setPixelPosition(getpixelposition(hUicontainer, true));
0260 end
0261 end
0262
0263 function configureComponent
0264     set(hUicontainer, 'DeleteFcn', {@containerDelete, hcomponent});
0265     addlistener(hcomponent, 'ObjectBeingDestroyed', {@componentDelete,
hUicontainer, parentIsFigure});
0266 end
0267
0268 end
0269
0270 function containerDelete(obj, evd, hc) %#ok - mlint
0271     if isa(handle(obj), 'uitoolbar')
0272         childHandles = getappdata(obj, 'childhandles');
0273         delete(childHandles);
0274     else
0275         if ishandle(hc)
0276             delete(hc);
0277         end
0278     end
0279 end
0280
0281 function componentDelete(obj, evd, hUicontainer, parentIsFigure) %#ok - mlint
0282 if (parentIsFigure)
0283     % This java component is always deleted before hUicontainer. It is
0284     % ensured by calling component deletion in function containerDelete.
0285     % hUicontainer becomes invalid when delete(hUicontainer) below is run.
0286     parent = ancestor(hUicontainer, 'figure');
0287     peer = get(parent, 'JavaFrame');
0288
0289     if any(ishandle(obj))
0290         removeobj = java(obj);
0291         if ~isempty(get(hUicontainer, 'FigureChild'))
0292             removeobj = get(hUicontainer, 'FigureChild');
0293         end

```

```

0294     peer.remove(removeobj);
0295 end
0296
0297 % delete container if it exists
0298 if any(ishandle(hUicontainer))
0299     delete(hUicontainer);
0300 end
0301 else
0302     parent = hUicontainer; % toolbar
0303     if ~ishandle(parent) || ~ishandle(obj)
0304         % The toolbar parent or the component has been deleted. Bail out.
0305         % Toolbar clears all javacomponents after itself.
0306         return;
0307     end
0308
0309     peer = get(parent, 'JavaContainer');
0310     if ~isempty(peer)
0311         peer.remove(java(obj));
0312     end
0313 end
0314 end
0315
0316
0317 function str=usage
0318 str = [sprintf('\n') ...
0319     'WARNING: This feature is not supported in MATLAB and the API and ' sprintf
0320     ('\n') ...
0321     'functionality may change in a future release. ' sprintf('\n') ...
0322 ];
0323 %{
0324     'usage: javacomponent(hJavaComponent, position, parent) ' sprintf('\n') ...
0325     '- position can be [left bottom width height] in pixels ' sprintf('\n') ...
0326     ' or the string North, South, East, or West' sprintf('\n') ...
0327     '- parent can be a figure or a uicontrol handle.'];
0328 %}
0329 end

```

[Home](#) > [classes](#) > [@time](#) > display.m

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 1.8 2008/01/14 20:03:52 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

```
0001 function txt = display(tt, varargin)
0002 % DISPLAY overloads display functionality for time objects.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY overloads display functionality for time objects.
0007 %
0008 % CALL:          txt  = display(time)
0009 %
0010 % INPUT:         time - time object
0011 %
0012 % OUTPUT:        txt  - cell array with strings to display the time object
0013 %
0014 % VERSION:       $Id: display.m,v 1.8 2008/01/14 20:03:52 ingo Exp $
0015 %
0016 % HISTORY:       23-07-2007 Diepholz
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: display.m,v 1.8 2008/01/14 20:03:52 ingo Exp $';
0022
0023 % Check if this is a call for parameters
0024 if nargin == 2
0025     if isa(tt, 'time') && ischar(varargin{1})
0026         in = char(varargin{1});
0027         if strcmp(in, 'Params')
0028             txt = plist;
0029             return
0030         elseif strcmp(in, 'Version')
0031             txt = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 txt = {};
0038
0039 for ii = 1:numel(tt)
0040     t = tt(ii);
0041
0042     b_banner = sprintf('----- time %02d -----', ii);
0043     txt{end+1} = b_banner;
0044     txt{end+1} = ' ';
0045
0046     fields = fieldnames(t);
0047     max_length = length(fields{1});
0048     for jj = 2:length(fields)
0049         if length(fields{jj}) > max_length
0050             max_length = length(fields{jj});
0051         end
0052     end
0053
0054     for jj = 1:length(fields)
0055         field = fields{jj};

```

```

0056
0057     str_field = [];
0058     str_field(1:max_length-length(field)) = ' ';
0059     str_field = [field str_field];
0060
0061     % Add before the fields 'created' and 'version' a blank line
0062     if strcmp(field, 'created')
0063         txt{end+1} = ' ';
0064     end
0065
0066     % Display: Number
0067     if isnumeric(t.(field))
0068         txt{end+1} = sprintf ('%s: %s',str_field, num2str(t.(field)));
0069
0070     % Display: Strings
0071     elseif ischar(t.(field))
0072         txt{end+1} = sprintf ('%s: %s',str_field, t.(field));
0073
0074     % Display: Logicals
0075     elseif islogical(t.(field))
0076         if t.(field) == true
0077             txt{end+1} = sprintf ('%s: true',str_field);
0078         else
0079             txt{end+1} = sprintf ('%s: false',str_field);
0080         end
0081
0082     % Display: Objects
0083     elseif isobject(t.(field))
0084         if isa(t.(field), 'timeformat')
0085             txt{end+1} = sprintf ('%s: %s',str_field, char(t.(field)));
0086         else
0087             txt{end+1} = sprintf ('%s: %s class',str_field, class(t.(field)));
0088         end
0089
0090     % Display: Java Objects
0091     elseif isjava(t.(field))
0092         if strcmp(class(t.(field)), 'sun.util.calendar.ZoneInfo')
0093             EE = java.text.SimpleDateFormat('Z');
0094             EE.setTimeZone(t.(field))
0095             zone_id = t.(field).getID;
0096             if strcmp(zone_id, 'UTC')
0097                 txt{end+1} = sprintf ('%s: %s',str_field, char(t.(field).getID));
0098             else
0099                 txt{end+1} = sprintf ('%s: %s (UTC%s)',str_field, char(t.(field).
0100 getID), char(EE.format(t.utc_epoch_milli)));
0101             end
0102         else
0103             error ('### Unknown java class %s', class(t.(field)))
0104         end
0105     else
0106         error ('### Please define the output for this property %s', field)

```



```
0107     end
0108
0109     end
0110
0111
0112     e_banner(1:length(b_banner)) = '-';
0113     txt{end+1} = e_banner;
0114
0115     txt{end+1} = ' ';
0116     txt{end+1} = ' ';
0117
0118 end
0119
0120 if nargout == 0
0121     for ii=1:length(txt)
0122         disp(txt{ii});
0123     end
0124 end
0125
0126
```

[Home](#) > [m](#) > [helper](#) > ltpda_label.m

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

```
0001 function s = ltpda_label(si)
0002 % LTPDA_LABEL makes the input string into a suitable string for using on plots.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

0005 %
0006 % DESCRIPTION: LTPDA_LABEL makes the input string into a suitable string for
0007 %                using on plots.
0008 %
0009 % CALL:         s = ltpda_label(sin)
0010 %
0011 % INPUTS:       sin - input string
0012 %
0013 % OUTPUTS:      s   - suitable string for the plot functions
0014 %
0015 % VERSION: $Id: ltpda_label.m,v 1.2 2007/07/13 12:17:39 ingo Exp $
0016 %
0017 % HISTORY: 11-02-2007 M Hewitson
0018 %          Creation
0019 %
0020 %%%%%%%%%%%
0021
0022 s = strrep(si, '_', '\_');

```

[Home](#) > [classes](#) > [@pole](#) > save.m

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 1.1 2007/08/29 18:38:40 hewitson Exp \$

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

```
>> pl = save(pole, 'Params')
```

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE a pole object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for pole objects.
0008 %
0009 % CALL:          Save a pole object as a .mat file.
0010 %                >> save(p, 'blah.mat')
0011 %                Save a pole object as an XML file.
0012 %                >> save(p, 'blah.xml')
0013 %                Save a pole object to file specified in plist.
0014 %                >> save(p, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:      $Id: save.m,v 1.1 2007/08/29 18:38:40 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(pole, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'pole') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041 ALGONAME = mfilename;
0042 VERSION  = '$Id: save.m,v 1.1 2007/08/29 18:38:40 hewitson Exp $';

```

```
0043
0044 if nargin ~= 2
0045     help mfilename;
0046     error('### incorrect inputs');
0047 end
0048
0049 % get object
0050 obj = varargin{1};
0051 if isa(varargin{2}, 'plist')
0052     pl = varargin{2};
0053 elseif ischar(varargin{2})
0054     pl = plist(param('filename', varargin{2}));
0055 else
0056     help mfilename;
0057     error('### incorrect inputs');
0058 end
0059
0060 % Save object
0061 ltpda_saveobj(obj, pl);
0062
0063 % Get default params
0064 function plo = getDefaultPL()
0065
0066 disp('* creating default plist...');
0067 plo = plist();
0068 plo = append(plo, param('filename', 'object.xml'));
0069 disp('* done. ');
0070
0071
0072 % END
```

[Home](#) > [classes](#) > [@cdata](#) > xmladd.m

xmladd

PURPOSE

XMLADD add a cdata object to the given node of the input XML DOM

SYNOPSIS

function xml = xmladd(ds, xml, nodename, node)

DESCRIPTION

XMLADD add a cdata object to the given node of the input XML DOM object.

M Hewitson 14-02-07

\$Id: xmladd.m,v 1.3 2007/11/09 14:07:39 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [get](#) GET get cdata properties.
- [xml](#) XML convert a cdata object to an DOM XML object.
- [xmladd](#) XMLADD add a cdata object to the given node of the input XML DOM

This function is called by:

- [xml](#) XML convert a cdata object to an DOM XML object.
- [xmladd](#) XMLADD add a cdata object to the given node of the input XML DOM

SOURCE CODE

```
0001 function xml = xmladd(ds, xml, nodename, node)
0002
0003 % XMLADD add a cdata object to the given node of the input XML DOM
0004 % object.
0005 %
0006 % M Hewitson 14-02-07
0007 %
```

```

0008 % $Id: xmladd.m,v 1.3 2007/11/09 14:07:39 ingo Exp $
0009 %
0010
0011 el = xml.docNode.createElement(nodename);
0012
0013 for j=1:length(ds)
0014
0015     d = ds(j);
0016
0017     elc = xml.docNode.createElement('cdata');
0018
0019     % add name
0020     elcc = xml.docNode.createElement('name');
0021     elcc.appendChild(xml.docNode.createTextNode(d.name));
0022     elc.appendChild(elcc);
0023
0024     % add Xunits
0025     elcc = xml.docNode.createElement('xunits');
0026     elcc.appendChild(xml.docNode.createTextNode(d.xunits));
0027     elc.appendChild(elcc);
0028
0029     % add Yunits
0030     elcc = xml.docNode.createElement('yunits');
0031     elcc.appendChild(xml.docNode.createTextNode(d.yunits));
0032     elc.appendChild(elcc);
0033
0034     % add Version
0035     elcc = xml.docNode.createElement('version');
0036     elcc.appendChild(xml.docNode.createTextNode(d.version));
0037     elc.appendChild(elcc);
0038
0039     % add Created
0040     xml = xmladd(get(d, 'created'), xml, 'created', elc);
0041
0042     % add Tag
0043     elcc = xml.docNode.createElement('xset');
0044     x = get(d, 'x');
0045     istr = '';
0046     for j=1:length(x)
0047         istr = [istr ' ' x{j}];
0048     end
0049     elcc.appendChild(xml.docNode.createTextNode(istr));
0050     elc.appendChild(elcc);
0051
0052     % Add shape
0053     elcc = xml.docNode.createElement('shape');
0054     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(size(d.y))));
0055     elc.appendChild(elcc);
0056

```



```

0057 if isreal(d.y)
0058     % add data type
0059     elcc = xml.docNode.createElement('dtype');
0060     elcc.appendChild(xml.docNode.createTextNode('real'));
0061     elc.appendChild(elcc);
0062
0063     % add y
0064     elcc = xml.docNode.createElement('data');
0065     x = reshape(d.y, 1, numel(d.y));
0066     nmin = getappdata(0, 'xmlsetsize');
0067     while ~isempty(x)
0068         elccc = xml.docNode.createElement('yset');
0069         n = min(length(x), nmin);
0070         elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(x(1:n))));
0071         x = x(n+1:end);
0072         elcc.appendChild(elccc);
0073     end
0074     elc.appendChild(elcc);
0075 else
0076     % add data type
0077     elcc = xml.docNode.createElement('dtype');
0078     elcc.appendChild(xml.docNode.createTextNode('complex'));
0079     elc.appendChild(elcc);
0080
0081     % add y
0082     elcc = xml.docNode.createElement('data');
0083     x = reshape(d.y, 1, numel(d.y));
0084     nmin = getappdata(0, 'xmlsetsize');
0085     while ~isempty(x)
0086         n = min(length(x), nmin);
0087         elccc = xml.docNode.createElement('ryset');
0088         elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(real(x(1:
n))))));
0089         elcc.appendChild(elccc);
0090         elccc = xml.docNode.createElement('iyset');
0091         elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(imag(x(1:
n))))));
0092         elcc.appendChild(elccc);
0093         x = x(n+1:end);
0094     end
0095     elc.appendChild(elcc);
0096 end
0097
0098 el.appendChild(elc);
0099 end
0100
0101 node.appendChild(el);
0102
0103
0104 % END

```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@timeformat](#) > display.m

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 1.4 2008/01/15 17:36:36 ingo 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:

SOURCE CODE

```
0001 function txt = display(tt, varargin)
0002 % DISPLAY overloads display functionality for timeformat objects.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY overloads display functionality for timeformat objects.
0007 %
0008 % CALL:          txt = display(tf)
0009 %
0010 % INPUT:         tf - timeformat object
0011 %
0012 % OUTPUT:        txt - cell array with strings to display the timeformat object
0013 %
0014 % VERSION:       $Id: display.m,v 1.4 2008/01/15 17:36:36 ingo Exp $
0015 %
0016 % HISTORY:       23-07-2007 Diepholz
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: display.m,v 1.4 2008/01/15 17:36:36 ingo Exp $';
0022
0023 % Check if this is a call for parameters or for the cvs-version number
0024 if nargin == 2
0025     if isa(tt, 'timeformat') && ischar(varargin{1})
0026         in = char(varargin{1});
0027         if strcmp(in, 'Params')
0028             txt = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             txt = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 txt = {};
0038
0039 for ii = 1:numel(tt)
0040     t = tt(ii);
0041
0042     b_banner = sprintf('----- timeformat %02d -----', ii);
0043     txt{end+1} = b_banner;
0044     txt{end+1} = ' ';
0045
0046     fields = fieldnames(t);
0047     max_length = length(fields{1});
0048     for jj = 2:length(fields)
0049         if length(fields{jj}) > max_length
0050             max_length = length(fields{jj});
0051         end
0052     end
0053
0054     for jj = 1:length(fields)
0055         field = fields{jj};

```

```

0056
0057     str_field = [];
0058     str_field(1:max_length-length(field)) = ' ';
0059     str_field = [field str_field];
0060
0061     % Display: Number
0062     if isnumeric(t.(field))
0063         txt{end+1} = sprintf ('%s: %s',str_field, num2str(t.(field)));
0064
0065     % Display: Strings
0066     elseif ischar(t.(field))
0067         txt{end+1} = sprintf ('%s: %s',str_field, t.(field));
0068
0069     % Display: Logicals
0070     elseif islogical(t.(field))
0071         if t.(field) == true
0072             txt{end+1} = sprintf ('%s: true',str_field);
0073         else
0074             txt{end+1} = sprintf ('%s: false',str_field);
0075         end
0076
0077     % Display: Objects
0078     elseif isobject(t.(field))
0079         txt{end+1} = sprintf ('%s: %s class',str_field, class(t.(field)));
0080
0081     % Display: Java Objects
0082     elseif isjava(t.(field))
0083         error ('### Unknown java class %s', class(t.(field)))
0084
0085     else
0086         error ('### Please define the output for this property %s', field)
0087     end
0088
0089 end
0090
0091
0092 e_banner(1:length(b_banner)) = '-';
0093 txt{end+1} = e_banner;
0094
0095 txt{end+1} = ' ';
0096 txt{end+1} = ' ';
0097
0098 end
0099
0100 if nargout == 0
0101     for ii=1:length(txt)
0102         disp(txt{ii});
0103     end
0104 end
0105
0106

```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [sigproc](#) > [frequency_domain](#) > ltpda_lcohere.m

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]
Kmin - minimum number of averages [default: 10]
Jdes - number of spectral frequencies to compute [default: 1000]
Win - a specwin window object
Only the design parameters of the window object are used; the window is recomputed for each DFT length inside the lpsd_core algorithm. [default: Kaiser with -200dB PSL]
Nolap - desired overlap [default: taken from window]
Order - order of detrending.
-1 - no detrending
0 - subtract mean
1 - subtract linear fit
N - subtract fit of polynomial, order N

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

```
>> pl = ltpda_lcohere('Params')
```

M Hewitson 02-02-07

\$Id: ltpda_lcohere.m,v 1.8 2007/11/08 10:24:54 hewitson Exp \$

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [ltpda_compute_f](#) LTPDA_COMPUTE_F compute the frequency and resolution vectors for use with mlpsd.

This function is called by:

SUBFUNCTIONS

- [function lmin = findShortestVector\(as\)](#)
- [function fs = findFsMax\(as\)](#)
- [function plo = getDefaultPL\(\)](#)
- [function varargout = mlcohere\(varargin\)](#)

SOURCE CODE

```
0001 function varargout = ltpda_lcohere(varargin)
0002 % LTPDA_LCOHERE implement weighted coherence estimation computed on a log
0003 % frequency axis.
0004 %
0005 % LTPDA_LCOHERE implement weighted cross-power estimation computed on a
0006 % log frequency axis.
0007 %
0008 % Usage:
0009 %     >> txy = ltpda_lcohere(as)
0010 %     >> txy = ltpda_lcohere(as, pl)
0011 %     >> [txy, sig2] = ltpda_lcohere(as, pl) % returns variance estimates
0012 %
0013 % Inputs:
0014 %     as - array of analysis objects
0015 %     pl - parameter list (see below)
0016 %
0017 % Outputs:
0018 %     bs - array of analysis objects, one for each input
0019 %
```



```

0020 % Parameter list:
0021 % Kdes - desired number of averages [default: 100]
0022 % Kmin - minimum number of averages [default: 10]
0023 % Jdes - number of spectral frequencies to compute [default: 1000]
0024 % Win - a specwin window object
0025 % Only the design parameters of the window object are used; the
0026 % window is recomputed for each DFT length inside the lpsd_core
0027 % algorithm. [default: Kaiser with -200dB PSL]
0028 % Nolap - desired overlap [default: taken from window]
0029 % Order - order of detrending.
0030 % -1 - no detrending
0031 % 0 - subtract mean
0032 % 1 - subtract linear fit
0033 % N - subtract fit of polynomial, order N
0034 %
0035 % The following call returns a parameter list object that contains the
0036 % default parameter values:
0037 %
0038 % >> pl = ltpda_lcohere('Params')
0039 %
0040 %
0041 % M Hewitson 02-02-07
0042 %
0043 % $Id: ltpda_lcohere.m,v 1.8 2007/11/08 10:24:54 hewitson Exp $
0044 %
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046 %
0047 % Check if this is a call for parameters
0048 if nargin == 1
0049     in = char(varargin{1});
0050     if strcmp(in, 'Params')
0051         varargout{1} = getDefaultPL();
0052         return
0053     end
0054 end
0055
0056 % capture input variable names
0057 invars = {};
0058 as = [];
0059 ps = [];
0060 for j=1:nargin
0061     if isa(varargin{j}, 'ao')
0062         as = [as varargin{j}];
0063         invars = [invars cellstr(inputname(j))];
0064     end
0065     if isa(varargin{j}, 'plist')
0066         ps = [ps varargin{j}];
0067     end
0068 end
0069
0070 ALGONAME = mfilename;
0071 VERSION = '$Id: ltpda_lcohere.m,v 1.8 2007/11/08 10:24:54 hewitson Exp $';

```

```

0072
0073 % initialise output array
0074 bs = [];
0075
0076 % check plist
0077 if isempty(ps)
0078     pl = getDefaultPL();
0079 else
0080     pl = combine(ps, getDefaultPL());
0081 end
0082
0083 % Go through each input AO and build a matrix of signals
0084 na    = length(as);
0085
0086 %----- Resample all AOs
0087
0088 disp('*** Resampling AOs...');
0089 fsmax = findFsMax(as);
0090 fspl  = plist(param('fsout', fsmax));
0091 bs = [];
0092 for i=1:na
0093     a = as(i);
0094     % check this is a time-series object
0095     if ~isa(a.data, 'tsdata')
0096         error('### ltpda_lcohere requires tsdata (time-series) inputs.');
```

```

0097     end
0098     % Check Fs
0099     if a.data.fs ~= fsmax
0100         warning(sprintf('!!! Resampling AO %s/%s to %f Hz', a.name, a.data.name,
fsmax));
0101         a = resample(a, fspl);
0102     end
0103     bs = [bs a];
0104 end
0105 as = bs;
0106 clear bs;
0107 disp('*** Done.');
```

```

0108
0109 %----- Truncate all vectors
0110
0111 % Get shortest vector
0112 disp('*** Truncating all vectors...');
```

```

0113 lmin = findShortestVector(as);
0114 nsecs = lmin / fsmax;
0115 bs = [];
0116 for i=1:na
0117     a = as(i);
0118     if len(a) ~= lmin
0119         warning(sprintf('!!! Truncating AO %s/%s to %d secs', a.name, a.data.name,
nsecs));
0120         bs = [bs select(a, 1:lmin)];
0121     else
```

```

0122     bs = [bs a];
0123 end
0124 end
0125 as = bs;
0126 clear bs;
0127 disp('*** Done. ');
0128
0129
0130 %----- Build signal Matrix
0131 N      = len(as(1)); % length of first signal
0132 iS     = zeros(N,na);
0133 for j=1:na
0134     a = as(j);
0135     iS(:,j) = a.data.y;
0136 end
0137
0138 %----- check input parameters
0139
0140 % Averaging
0141 Kdes = find(pl, 'Kdes'); % desired averages
0142 Kmin = find(pl, 'Kmin'); % minimum averages
0143 Jdes = find(pl, 'Jdes'); % num desired spectral frequencies
0144 Win  = find(pl, 'Win');  % Window object
0145 Nlap = find(pl, 'Nlap'); % desired overlap
0146 if Nlap == -1
0147     % use window function rov
0148     Nlap = Win.rov/100;
0149     disp(sprintf('! Using recommended overlap of %d', Nlap))
0150     pl = pset(pl, 'Nlap', Nlap);
0151 end
0152 Order = find(pl, 'Order'); % desired overlap
0153
0154 % Debug
0155 debug = find(pl, 'DEBUG');
0156
0157 %----- Get frequency vector
0158 [f,r,m,L] = ltpda_compute_f(fsmax, lmin, Kdes, Kmin, Jdes, Nlap, debug);
0159
0160 %----- compute CPSD Estimates
0161
0162 if nargin == 2
0163     disp('*** Computing Coherence matrix with variance estimates')
0164     tic
0165     [Txy, ENBW, Sig] = mlcohere(iS, f, r, m, L, fsmax, Win, Order);
0166     toc
0167 else
0168     disp('*** Computing Coherence matrix')
0169     tic
0170     [Txy, ENBW] = mlcohere(iS, f, r, m, L, fsmax, Win, Order);
0171     toc
0172 end
0173

```

```

0174 %----- Build output Matrix of AOs
0175 disp('** Building output AO matrix')
0176 bs = cell(na);
0177 sig = cell(na);
0178 for j=1:na % Loop over input signal
0179     bi = as(j);
0180     for k=1:na % Loop over output signal
0181         bo = as(k);
0182         % create new output fsdata
0183         fs = fsdata(f, squeeze(Txy(j,k,:)), fsmax);
0184         fs = set(fs, 'name', sprintf('LCOHERE(%s->%s)', bi.data.name, bo.data.
name));
0185         fs = set(fs, 'yunits', [bo.data.yunits '/' bi.data.yunits]);
0186         fs = set(fs, 'enbw', squeeze(ENBW(j,k,:)));
0187         % create new output history
0188         h = history(ALGONAME, VERSION, pl, [bi.hist bo.hist]);
0189         if na == length(invars)
0190             h = set(h, 'invars', invars([j k]));
0191             nk = invars{k};
0192             nj = invars{j};
0193         else
0194             nk = bo.name;
0195             nj = bi.name;
0196         end
0197         % make output analysis object
0198         b = ao(fs, h);
0199         % set name
0200         b = setnh(b, 'name', sprintf('LCOHERE(%s->%s)', nj, nk));
0201         % add to outputs
0202         bs(j,k) = {b};
0203
0204         % Make variance outputs
0205         if nargout == 2
0206             % create new output fsdata
0207             fs = fsdata(f, squeeze(Sig(j,k,:)), fsmax);
0208             fs = set(fs, 'name', ['\sigma^2' sprintf('(%s->%s)', bi.data.name, bo.
data.name)]);
0209             fs = set(fs, 'yunits', [bo.data.yunits '/' bi.data.yunits]);
0210             fs = set(fs, 'enbw', squeeze(ENBW(j,k,:)));
0211             % create new output history
0212             h = history(ALGONAME, VERSION, pl, [bi.hist bo.hist]);
0213             if na == length(invars)
0214                 h = set(h, 'invars', invars{[j k]});
0215             end
0216             % make output analysis object
0217             b = ao(fs, h);
0218             % set name
0219             b = setnh(b, 'name', ['\sigma^2' sprintf('(%s->%s)', nj, nk)]);
0220             % add to outputs
0221             sig(j,k) = {b};
0222
0223     end

```

```

0224     end % End loop over output signal
0225 end % End loop over input signal
0226
0227 % now we have a cell matrix of AOs but
0228 % we want a normal matrix
0229 b = [bs{:}];
0230 bs = reshape(b, na, na);
0231 varargout{1} = bs;
0232 if nargout == 2
0233     b = [sig{:}];
0234     s = reshape(b, na, na);
0235     varargout{2} = s;
0236 end
0237 %-----
0238 % Returns the length of the shortest vector in samples
0239 function lmin = findShortestVector(as)
0240
0241 lmin = 1e20;
0242 for j=1:length(as)
0243     if len(as(j)) < lmin
0244         lmin = len(as(j));
0245     end
0246 end
0247
0248
0249 %-----
0250 % Returns the max Fs of a set of AOs
0251 function fs = findFsMax(as)
0252
0253 fs = 0;
0254 for j=1:length(as)
0255     a = as(j);
0256     if a.data.fs > fs
0257         fs = a.data.fs;
0258     end
0259 end
0260
0261 %-----
0262 % Get default params
0263 function plo = getDefaultPL()
0264
0265 disp('* creating default plist...');
0266 plo = plist();
0267 plo = append(plo, param('Kdes', 100));
0268 plo = append(plo, param('Jdes', 1000));
0269 plo = append(plo, param('Kmin', 2));
0270 plo = append(plo, param('Win', specwin('Kaiser', 1, 200)));
0271 plo = append(plo, param('Nolap', -1));
0272 plo = append(plo, param('Order', 0));
0273 disp('* done.');
```

```

0276 %-----
0277 % Compute all TF combinations
0278 function varargout = mlcohere(varargin)
0279
0280 % MLCOHERE
0281 %
0282 % Txy = mlcohere(X,f,r,m,L,fs,win)
0283 %
0284 % M Hewitson 19-05-07
0285 %
0286 %
0287
0288 % Get inputs
0289 X      = varargin{1};
0290 f      = varargin{2};
0291 r      = varargin{3};
0292 m      = varargin{4};
0293 L      = varargin{5};
0294 fs     = varargin{6};
0295 win    = varargin{7};
0296 order  = varargin{8};
0297
0298 % do we need to compute variance?
0299 if nargin == 2
0300     computeVariance = 1;
0301 else
0302     computeVariance = 0;
0303 end
0304
0305 twopi    = 2.0*pi;
0306 twopiofs = 2.0*pi/(1.0*fs);
0307
0308 si       = size(X);
0309 nx       = si(1);
0310 nc       = si(2);
0311 nf       = length(f);
0312 Txy      = zeros(nc,nc,nf);
0313 Sig      = zeros(nc,nc,nf);
0314 ENBW     = zeros(nc,nc,nf);
0315
0316
0317 switch order
0318     case -1
0319         detrender = '';
0320     case 0
0321         detrender = 'xi = xi - mean(xi);xo = xo - mean(xo);';
0322     case 1
0323         detrender = 'xi = detrend(xi);xo = detrend(xo);';
0324     otherwise
0325         detrender = 'xi = polydetrend(xi, order); xo = polydetrend(xo, order);';
0326 end
0327

```

```

0328 disp_each = round(nf/100)*10;
0329
0330 for fi=int32(1:nf)
0331
0332     % compute DFT exponent and window
0333     l = L(fi);
0334     switch win.name
0335         case 'Kaiser'
0336             win = specwin(win.name, l, win.psl1);
0337         otherwise
0338             win = specwin(win.name, l);
0339     end
0340
0341     p      = 1i * twopi * m(fi)/l.*[0:l-1];
0342     C      = win.win .* exp(p);
0343     Xolap  = (1-win.rov/100);
0344
0345     if mod(fi,disp_each) == 0 || fi == 1 || fi == nf
0346         disp(sprintf('++ computing frequency %03d of %03d: %f Hz', fi, nf, f(fi)));
0347     end
0348
0349     %----- Count segments
0350     nsegs = 0;
0351     start = 1;
0352     lm1   = l-1;
0353     while start + l - 1 <= nx
0354         start = floor(start + l*Xolap);
0355         nsegs = nsegs + 1;
0356     end
0357
0358     % now allocate indices for segments
0359 %     idxs = repmat(struct('idx', zeros(1,l)), 1, nsegs);
0360     idxs = zeros(nsegs,l, 'uint32');
0361     nsegs = 0;
0362     start = 1;
0363     while start + l - 1 <= nx
0364         idxs(1+nsegs,:) = start:start+lm1;
0365         start = floor(start + l*Xolap);
0366         nsegs = nsegs + 1;
0367     end
0368     Sxy = zeros(nsegs,1);
0369     i2  = zeros(nsegs,1);
0370     xi  = zeros(l,1);
0371     xo  = zeros(l,1);
0372
0373     %----- Process the channels
0374
0375     % Loop over input channels
0376     for j=1:nc
0377         % loop over output channels
0378         for k=1:nc
0379             % compute lower triangle of matrix

```

```

0380     if j<k
0381
0382         % do segments
0383         Sxx      = 0;
0384         Syy      = 0;
0385         Sxy      = 0;
0386         i2(:)    = 0;
0387
0388         % Loop over segments
0389         for s=1:nsegs
0390             % get segments
0391             xi(:) = X(idxs(s,:),j);
0392             xo(:) = X(idxs(s,:),k);
0393
0394             % detrend segments
0395             eval(detrender);
0396
0397             % compute DFT
0398             ti      = C*xi;
0399             to      = C*xo;
0400             cto     = conj(to);
0401             Sxy     = Sxy + ti.*cto;
0402             Sxx     = Sxx + ti.*conj(ti);
0403             Syy     = Syy + to.*cto;
0404
0405         end % End segment loop
0406         disp(['-processed ' num2str(nsegs) ' segments']);
0407
0408         Sxy = Sxy / nsegs;
0409         Sxx = Sxx / nsegs;
0410         Syy = Syy / nsegs;
0411
0412         S1      = win.ws;
0413         S12     = S1*S1;
0414         S2      = win.ws2;
0415         ENBW(j,k,fi) = fs*S2/S12;
0416         Txy(j,k,fi) = (abs(Sxy)^2)/(Sxx*Syy);
0417
0418         %----- compute variance
0419         if computeVariance
0420             A      = 0;
0421             scale = fs*S2;
0422             for s=1:nsegs
0423                 S = (Sxy(s)/scale-Txy(j,k,fi))^2;
0424                 A = A + S * i2(s);
0425             end
0426             Sig(j,k,fi) = A/sum(i2);
0427         end % End compute variance
0428
0429     end % End if j<k
0430 end % End loop over output channels
0431 end % End loop over input channels

```



```
0432 end % End loop over frequencies
0433
0434 % Fill the holes
0435 ao0nes = ones(1,nf);
0436 for j=1:nc
0437     for k=1:nc
0438         if j == k % the answer is 1
0439             Txy(j,k,:) = ao0nes;
0440         end
0441         if j>k
0442             Txy(j,k,:) = Txy(k,j,:);
0443         if computeVariance
0444             Sig(j,k,:) = Sig(k,j,:);
0445         end
0446     end
0447 end
0448 end
0449
0450
0451 varargout{1} = Txy;
0452 if nargout == 2
0453     varargout{2} = ENBW;
0454 elseif nargout == 3
0455     varargout{3} = Sig;
0456     varargout{2} = ENBW;
0457 end
0458
0459
0460 % END
```

[Home](#) > [classes](#) > [@provenance](#) > save.m

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 1.1 2007/08/29 18:39:05 hewitson Exp \$

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

```
>> pl = save(provenance, 'Params')
```

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE a provenance object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for provenance objects.
0008 %
0009 % CALL:          Save a provenance object as a .mat file.
0010 %                >> save(p, 'blah.mat')
0011 %                Save a provenance object as an XML file.
0012 %                >> save(p, 'blah.xml')
0013 %                Save a provenance object to file specified in plist.
0014 %                >> save(p, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:      $Id: save.m,v 1.1 2007/08/29 18:39:05 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(provenance, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'provenance') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041 ALGONAME = mfilename;
0042 VERSION  = '$Id: save.m,v 1.1 2007/08/29 18:39:05 hewitson Exp $';

```

```
0043
0044 if nargin ~= 2
0045     help mfilename;
0046     error('### incorrect inputs');
0047 end
0048
0049 % get object
0050 obj = varargin{1};
0051 if isa(varargin{2}, 'plist')
0052     pl = varargin{2};
0053 elseif ischar(varargin{2})
0054     pl = plist(param('filename', varargin{2}));
0055 else
0056     help mfilename;
0057     error('### incorrect inputs');
0058 end
0059
0060 % Save object
0061 ltpda_saveobj(obj, pl);
0062
0063 % Get default params
0064 function plo = getDefaultPL()
0065
0066 disp('* creating default plist...');
0067 plo = plist();
0068 plo = append(plo, param('filename', 'object.xml'));
0069 disp('* done. ');
0070
0071
0072 % END
```

[Home](#) > [classes](#) > [@fsdata](#) > xmladd.m

xmladd

PURPOSE

XMLADD add a fsdata object to the given node of the input XML DOM

SYNOPSIS

function xml = xmladd(ds, xml, nodename, node)

DESCRIPTION

XMLADD add a fsdata object to the given node of the input XML DOM object.

M Hewitson 14-02-07

\$Id: xmladd.m,v 1.3 2007/11/07 15:57:26 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [get](#) GET get a fsdata property.
- [xml](#) XML convert an fsdata object to an DOM XML object.
- [xmladd](#) XMLADD add a fsdata object to the given node of the input XML DOM

This function is called by:

- [xml](#) XML convert an fsdata object to an DOM XML object.
- [xmladd](#) XMLADD add a fsdata object to the given node of the input XML DOM

SOURCE CODE

```
0001 function xml = xmladd(ds, xml, nodename, node)
0002
0003 % XMLADD add a fsdata object to the given node of the input XML DOM
0004 % object.
0005 %
0006 % M Hewitson 14-02-07
0007 %
```

```

0008 % $Id: xmladd.m,v 1.3 2007/11/07 15:57:26 ingo Exp $
0009 %
0010
0011 el = xml.docNode.createElement(nodename);
0012
0013 for j=1:length(ds)
0014
0015     d = ds(j);
0016
0017     elc = xml.docNode.createElement('fsdata');
0018
0019     % add name
0020     elcc = xml.docNode.createElement('name');
0021     elcc.appendChild(xml.docNode.createTextNode(d.name));
0022     elc.appendChild(elcc);
0023
0024     % add fs
0025     elcc = xml.docNode.createElement('fs');
0026     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(d.fs)));
0027     elc.appendChild(elcc);
0028
0029     % add enbw
0030     elcc = xml.docNode.createElement('enbw');
0031     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(d.enbw)));
0032     elc.appendChild(elcc);
0033
0034     % add Navs
0035     elcc = xml.docNode.createElement('navs');
0036     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(d.navs)));
0037     elc.appendChild(elcc);
0038
0039     % add Xunits
0040     elcc = xml.docNode.createElement('xunits');
0041     elcc.appendChild(xml.docNode.createTextNode(d.xunits));
0042     elc.appendChild(elcc);
0043
0044     % add Yunits
0045     elcc = xml.docNode.createElement('yunits');
0046     elcc.appendChild(xml.docNode.createTextNode(d.yunits));
0047     elc.appendChild(elcc);
0048
0049     % add Version
0050     elcc = xml.docNode.createElement('version');
0051     elcc.appendChild(xml.docNode.createTextNode(d.version));
0052     elc.appendChild(elcc);
0053
0054     % add Created
0055     xml = xmladd(get(d, 'created'), xml, 'created', elc);
0056
0057     if isreal(d.y)

```

```

0058 % add data type
0059 elcc = xml.docNode.createElement('dtype');
0060 elcc.appendChild(xml.docNode.createTextNode('real'));
0061 elc.appendChild(elcc);
0062
0063 % add real y
0064 elcc = xml.docNode.createElement('data');
0065 x = d.y;
0066 nmin = getappdata(0, 'xmlsetsize');
0067 while ~isempty(x)
0068     elccc = xml.docNode.createElement('yset');
0069     n = min(length(x), nmin);
0070     elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(x(1:n))));
0071     x = x(n+1:end);
0072     elcc.appendChild(elccc);
0073 end
0074 % add x
0075 x = d.x;
0076 nmin = getappdata(0, 'xmlsetsize');
0077 while ~isempty(x)
0078     elccc = xml.docNode.createElement('xset');
0079     n = min(length(x), nmin);
0080     elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(x(1:n))));
0081     x = x(n+1:end);
0082     elcc.appendChild(elccc);
0083 end
0084 elc.appendChild(elcc);
0085 else
0086 % add data type
0087 elcc = xml.docNode.createElement('dtype');
0088 elcc.appendChild(xml.docNode.createTextNode('complex'));
0089 elc.appendChild(elcc);
0090
0091 % add real y
0092 elcc = xml.docNode.createElement('data');
0093 x = d.y;
0094 nmin = getappdata(0, 'xmlsetsize');
0095 while ~isempty(x)
0096     n = min(length(x), nmin);
0097     elccc = xml.docNode.createElement('ryset');
0098     elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(real(x(1:
n))))));
0099     elcc.appendChild(elccc);
0100     elccc = xml.docNode.createElement('iyset');
0101     elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(imag(x(1:
n))))));
0102     elcc.appendChild(elccc);
0103     x = x(n+1:end);
0104 end
0105 % add x

```

```
0106     x      = d.x;
0107     nmin = getappdata(0, 'xmlsetsize');
0108     while ~isempty(x)
0109         elccc = xml.docNode.createElement('xset');
0110         n = min(length(x), nmin);
0111         elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(x(1:n))));
0112         x = x(n+1:end);
0113         elcc.appendChild(elccc);
0114     end
0115     elc.appendChild(elcc);
0116 end
0117
0118 el.appendChild(elc);
0119 end
0120
0121 node.appendChild(el);
0122
0123
0124 % END
```


[Home](#) > [classes](#) > [@timespan](#) > display.m

display

PURPOSE

DISPLAY overloads display functionality for timespan objects.

SYNOPSIS

function txt = display(t_span)

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 1.4 2008/01/08 16:39:12 ingo Exp \$

HISTORY: 23-07-2007 Diepholz
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a timespan object into a string.

This function is called by:

SOURCE CODE

```
0001 function txt = display(t_span)
0002 % DISPLAY overloads display functionality for timespan objects.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY overloads display functionality for timespan objects.
0007 %
0008 % CALL:          txt = display(ts)
0009 %
0010 % INPUT:         ts  - timespan object
0011 %
0012 % OUTPUT:        txt - cell array with strings to display the timespan object
0013 %
0014 % VERSION:       $Id: display.m,v 1.4 2008/01/08 16:39:12 ingo Exp $
0015 %
0016 % HISTORY:       23-07-2007 Diepholz
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 txt = {};
0022
0023 for ii = 1:numel(t_span)
0024     t = t_span(ii);
0025
0026     b_banner = sprintf('----- timespan %02d -----', ii);
0027     txt{end+1} = b_banner;
0028     txt{end+1} = ' ';
0029
0030     fields = fieldnames(t);
0031     max_length = length(fields{1});
0032     for jj = 2:length(fields)
0033         if length(fields{jj}) > max_length
0034             max_length = length(fields{jj});
0035         end
0036     end
0037
0038     for jj = 1:length(fields)
0039         field = fields{jj};
0040
0041         str_field = [];
0042         str_field(1:max_length-length(field)) = ' ';
0043         str_field = [field str_field];
0044
0045         % Add before the fields 'created' and 'version' a blank line
0046         if strcmp(field, 'created')
0047             txt{end+1} = ' ';
0048         end
0049
0050         % Display: Number
0051         if isnumeric(t.(field))
0052             txt{end+1} = sprintf('%s: %s', str_field, num2str(t.(field)));
0053
0054         % Display: Strings
0055         elseif ischar(t.(field))

```

```

0056     txt{end+1} = sprintf ('%s: %s',str_field, t.(field));
0057
0058     % Display: Logicals
0059     elseif islogical(t.(field))
0060         if t.(field) == true
0061             txt{end+1} = sprintf ('%s: true',str_field);
0062         else
0063             txt{end+1} = sprintf ('%s: false',str_field);
0064         end
0065
0066     % Display: Objects
0067     elseif isobject(t.(field))
0068         if isa(t.(field), 'time') || isa(t.(field), 'timeformat')
0069             txt{end+1} = sprintf ('%s: %s',str_field, char(t.(field)));
0070         else
0071             txt{end+1} = sprintf ('%s: %s class',str_field, class(t.(field)));
0072         end
0073
0074     % Display: Java Objects
0075     elseif isjava(t.(field))
0076         if strcmp(class(t.(field)), 'sun.util.calendar.ZoneInfo')
0077             EE = java.text.SimpleDateFormat('Z');
0078             EE.setTimeZone(t.(field))
0079             zone_id = t.(field).getID;
0080             if strcmp(zone_id, 'UTC')
0081                 txt{end+1} = sprintf ('%s: %s',str_field, char(t.(field).getID));
0082             else
0083                 txt{end+1} = sprintf ('%s: %s (UTC%s)',str_field, char(t.(field).
0084 getID), char(EE.format(t_span.start.utc_epoch_milli)));
0085             end
0086         else
0087             error ('### Unknown java class %s', class(t.(field)))
0088         end
0089     else
0090         error ('### Please define the output for this property %s', field)
0091     end
0092
0093 end
0094
0095
0096 e_banner(1:length(b_banner)) = '-';
0097 txt{end+1} = e_banner;
0098
0099 txt{end+1} = ' ';
0100 txt{end+1} = ' ';
0101
0102 end
0103
0104 if nargin == 0
0105     for ii=1:length(txt)
0106         disp(txt{ii});

```

```
0107    end
0108  end
0109
0110
0111
0112
```

[Home](#) > [m](#) > [sigproc](#) > [frequency_domain](#) > ltpda_lcpsd.m

ltpda_lcpsd

PURPOSE

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

SYNOPSIS

function varargout = ltpda_lcpsd(varargin)

DESCRIPTION

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

%%%

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

Usage:

```
>> txy = ltpda_lcpsd(as)
>> txy = ltpda_lcpsd(as, pl)
>> [txy, sig2] = ltpda_lcpsd(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]
Kmin - minimum number of averages [default: 10]
Jdes - number of spectral frequencies to compute [default: fs/4]
Win - a specwin window object
Only the design parameters of the window object are used; the window is recomputed for each DFT length inside the lpsd_core algorithm. [default: Kaiser with -200dB PSL]
Nolap - desired overlap [default: taken from window]
Order - order of detrending.
-1 - no detrending
0 - subtract mean
1 - subtract linear fit
N - subtract fit of polynomial, order N

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

```
>> pl = ltpda_lcpsd('Params')
```

M Hewitson 02-02-07

\$Id: ltpda_lcpsd.m,v 1.5 2007/11/08 10:24:54 hewitson Exp \$

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [ltpda_compute_f](#) LTPDA_COMPUTE_F compute the frequency and resolution vectors for use with mlpsd.

This function is called by:

SUBFUNCTIONS

- [function lmin = findShortestVector\(as\)](#)
- [function fs = findFsMax\(as\)](#)
- [function plo = getDefaultPL\(\)](#)
- [function varargout = mlcpsd\(varargin\)](#)

SOURCE CODE

```
0001 function varargout = ltpda_lcpsd(varargin)
0002 % LTPDA_LCPSTD implement weighted cross-power estimation computed on a log
0003 % frequency axis.
0004 %
0005 % LTPDA_LCPSTD implement weighted cross-power estimation computed on a
0006 % log frequency axis.
0007 %
0008 % Usage:
0009 %     >> txy = ltpda_lcpsd(as)
0010 %     >> txy = ltpda_lcpsd(as, pl)
0011 %     >> [txy, sig2] = ltpda_lcpsd(as, pl) % returns variance estimates
0012 %
0013 %     Inputs:
0014 %         as - array of analysis objects
0015 %         pl - parameter list (see below)
0016 %
0017 %     Outputs:
0018 %         bs - array of analysis objects, one for each input
0019 %
```

```

0020 % Parameter list:
0021 % Kdes - desired number of averages [default: 100]
0022 % Kmin - minimum number of averages [default: 10]
0023 % Jdes - number of spectral frequencies to compute [default: fs/4]
0024 % Win - a specwin window object
0025 % Only the design parameters of the window object are used; the
0026 % window is recomputed for each DFT length inside the lpsd_core
0027 % algorithm. [default: Kaiser with -200dB PSLL]
0028 % Nolap - desired overlap [default: taken from window]
0029 % Order - order of detrending.
0030 % -1 - no detrending
0031 % 0 - subtract mean
0032 % 1 - subtract linear fit
0033 % N - subtract fit of polynomial, order N
0034 %
0035 % The following call returns a parameter list object that contains the
0036 % default parameter values:
0037 %
0038 % >> pl = ltpda_lcpsd('Params')
0039 %
0040 %
0041 % M Hewitson 02-02-07
0042 %
0043 % $Id: ltpda_lcpsd.m,v 1.5 2007/11/08 10:24:54 hewitson Exp $
0044 %
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046 %
0047 % Check if this is a call for parameters
0048 if nargin == 1
0049     in = char(varargin{1});
0050     if strcmp(in, 'Params')
0051         varargout{1} = getDefaultPL();
0052         return
0053     end
0054 end
0055
0056 % capture input variable names
0057 invars = {};
0058 as = [];
0059 ps = [];
0060 for j=1:nargin
0061     invars = [invars cellstr(inputname(j))];
0062     if isa(varargin{j}, 'ao')
0063         as = [as varargin{j}];
0064     end
0065     if isa(varargin{j}, 'plist')
0066         ps = [ps varargin{j}];
0067     end
0068 end
0069
0070 ALGONAME = mfilename;
0071 VERSION = '$Id: ltpda_lcpsd.m,v 1.5 2007/11/08 10:24:54 hewitson Exp $';

```

```

0072
0073 % initialise output array
0074 bs = [];
0075
0076 % check plist
0077 if isempty(ps)
0078     pl = getDefaultPL();
0079 else
0080     pl = combine(ps, getDefaultPL());
0081 end
0082
0083 % Go through each input AO and build a matrix of signals
0084 na    = length(as);
0085
0086 %----- Resample all AOs
0087
0088 disp('*** Resampling AOs...');
0089 fsmax = findFsMax(as);
0090 fspl  = plist(param('fsout', fsmax));
0091 bs = [];
0092 for i=1:na
0093     a = as(i);
0094     % check this is a time-series object
0095     if ~isa(a.data, 'tsdata')
0096         error('### ltpda_lcpsd requires tsdata (time-series) inputs.');
```

```

0097     end
0098     % Check Fs
0099     if a.data.fs ~= fsmax
0100         warning(sprintf('!!! Resampling AO %s/%s to %f Hz', a.name, a.data.name,
fsmax));
0101         a = resample(a, fspl);
0102     end
0103     bs = [bs a];
0104 end
0105 as = bs;
0106 clear bs;
0107 disp('*** Done.');
```

```

0108
0109 %----- Truncate all vectors
0110
0111 % Get shortest vector
0112 disp('*** Truncating all vectors...');
```

```

0113 lmin = findShortestVector(as);
0114 nsecs = lmin / fsmax;
0115 bs = [];
0116 for i=1:na
0117     a = as(i);
0118     if len(a) ~= lmin
0119         warning(sprintf('!!! Truncating AO %s/%s to %d secs', a.name, a.data.name,
nsecs));
0120         bs = [bs select(a, 1:lmin)];
0121     else
```



```

0122     bs = [bs a];
0123     end
0124 end
0125 as = bs;
0126 clear bs;
0127 disp('*** Done. ');
0128
0129
0130 %----- Build signal Matrix
0131 N      = len(as(1)); % length of first signal
0132 iS     = zeros(na, N);
0133 for j=1:na
0134     a = as(j);
0135     iS(j,:) = a.data.y;
0136 end
0137
0138 %----- check input parameters
0139
0140 % Averaging
0141 Kdes = find(pl, 'Kdes'); % desired averages
0142 Kmin = find(pl, 'Kmin'); % minimum averages
0143 Jdes = find(pl, 'Jdes'); % num desired spectral frequencies
0144 if Jdes == -1
0145     Jdes = round(fsmax/4);
0146     disp(sprintf('! Using default Jdes of %d', Jdes))
0147     pl = pset(pl, 'Jdes', Jdes);
0148 end
0149 Win  = find(pl, 'Win'); % Window object
0150 Nlap = find(pl, 'Nlap'); % desired overlap
0151 if Nlap == -1
0152     % use window function rov
0153     Nlap = Win.rov/100;
0154     disp(sprintf('! Using recommended overlap of %d', Nlap))
0155     pl = pset(pl, 'Nlap', Nlap);
0156 end
0157 Order = find(pl, 'Order'); % desired overlap
0158 if isempty(Order)
0159     Order = 0;
0160     disp(sprintf('! Using default detrending order of %d', Order))
0161     pl = append(pl, param('Order', Order));
0162 end
0163
0164
0165 %----- Get frequency vector
0166 [f,r,m,L] = ltpda_compute_f(fsmax, lmin, Kdes, Kmin, Jdes, Nlap);
0167
0168 %----- compute CPSD Estimates
0169
0170 if nargout == 2
0171     disp('** Computing CPSD matrix with variance estimates')
0172     tic
0173     [Txy, ENBW, Sig] = mlcpsd(iS, f, r, m, L, fsmax, Win, Order);

```

```

0174     toc
0175 else
0176     disp('** Computing CPSD matrix')
0177     tic
0178     [Txy, ENBW] = mlcpsd(iS, f, r, m, L, fsmax, Win, Order);
0179     toc
0180 end
0181
0182 %----- Build output Matrix of AOs
0183 disp('** Building output AO matrix')
0184 bs = cell(na);
0185 sig = cell(na);
0186 for j=1:na % Loop over input signal
0187     bi = as(j);
0188     for k=1:na % Loop over output signal
0189         bo = as(k);
0190         % create new output fsdata
0191         fs = fsdata(f, squeeze(Txy(j,k,:)), fsmax);
0192         fs = set(fs, 'name', sprintf('LCPSD(%s->%s)', bi.data.name, bo.data.name));
0193         fs = set(fs, 'yunits', [bo.data.yunits '/' bi.data.yunits]);
0194         fs = set(fs, 'enbw', squeeze(ENBW(j,k,:)));
0195         % create new output history
0196         h = history(ALGONAME, VERSION, pl, [bi.hist bo.hist]);
0197         h = set(h, 'invars', invars([j k]));
0198         % make output analysis object
0199         b = ao(fs, h);
0200         % set name
0201         if isempty(invars{k})
0202             nk = bo.name;
0203         else
0204             nk = invars{k};
0205         end
0206         if isempty(invars{j})
0207             nj = bi.name;
0208         else
0209             nj = invars{j};
0210         end
0211         b = setnh(b, 'name', sprintf('LCPSD(%s->%s)', nj, nk));
0212         % add to outputs
0213         bs(j,k) = {b};
0214
0215         % Make variance outputs
0216         if nargout == 2
0217             % create new output fsdata
0218             fs = fsdata(f, squeeze(Sig(j,k,:)), fsmax);
0219             fs = set(fs, 'name', ['\sigma^2' sprintf('(%s->%s)', bi.data.name, bo.
data.name)]);
0220             fs = set(fs, 'yunits', [bo.data.yunits '/' bi.data.yunits]);
0221             fs = set(fs, 'enbw', squeeze(ENBW(j,k,:)));
0222             % create new output history
0223             h = history(ALGONAME, VERSION, pl, [bi.hist bo.hist]);
0224             h = set(h, 'invars', invars([j k]));

```

```

0225     % make output analysis object
0226     b = ao(fs, h);
0227     % set name
0228     if isempty(invars{k})
0229         nk = bo.name;
0230     else
0231         nk = invars{k};
0232     end
0233     if isempty(invars{j})
0234         nj = bi.name;
0235     else
0236         nj = invars{j};
0237     end
0238     b = setnh(b, 'name', ['\sigma^2' sprintf('(%s->%s)', nj, nk)]);
0239     % add to outputs
0240     sig(j,k) = {b};
0241
0242 end
0243 end % End loop over output signal
0244 end % End loop over input signal
0245
0246 % now we have a cell matrix of AOs but
0247 % we want a normal matrix
0248 b = [bs{:}];
0249 bs = reshape(b, na, na);
0250 varargout{1} = bs;
0251 if nargin == 2
0252     b = [sig{:}];
0253     s = reshape(b, na, na);
0254     varargout{2} = s;
0255 end
0256 %-----
0257 % Returns the length of the shortest vector in samples
0258 function lmin = findShortestVector(as)
0259
0260 lmin = 1e20;
0261 for j=1:length(as)
0262     if len(as(j)) < lmin
0263         lmin = len(as(j));
0264     end
0265 end
0266
0267
0268 %-----
0269 % Returns the max Fs of a set of AOs
0270 function fs = findFsMax(as)
0271
0272 fs = 0;
0273 for j=1:length(as)
0274     a = as(j);
0275     if a.data.fs > fs
0276         fs = a.data.fs;

```

```

0277     end
0278 end
0279
0280 %-----
0281 % Get default params
0282 function plo = getDefaultPL()
0283
0284 disp('* creating default plist...');
0285 plo = plist();
0286 plo = append(plo, param('Kdes', 100));
0287 plo = append(plo, param('Jdes', 1000));
0288 plo = append(plo, param('Kmin', 2));
0289 plo = append(plo, param('Win', specwin('Kaiser', 1, 150)));
0290 plo = append(plo, param('Nolap', -1));
0291 plo = append(plo, param('Order', 0));
0292 disp('* done. ');
0293
0294
0295 %-----
0296 % Compute all TF combinations
0297 function varargout = mlcpsd(varargin)
0298
0299 % MLCPSD
0300 %
0301 % Txy = mlcpsd(X,f,r,m,L,fs,win)
0302 %
0303 % M Hewitson 19-05-07
0304 %
0305 %
0306
0307 % Get inputs
0308 X      = varargin{1};
0309 f      = varargin{2};
0310 r      = varargin{3};
0311 m      = varargin{4};
0312 L      = varargin{5};
0313 fs     = varargin{6};
0314 win    = varargin{7};
0315 order  = varargin{8};
0316
0317 % do we need to compute variance?
0318 if nargin == 2
0319     computeVariance = 1;
0320 else
0321     computeVariance = 0;
0322 end
0323
0324 twopi    = 2.0*pi;
0325 twopiofs = 2.0*pi/(1.0*fs);
0326
0327 si       = size(X);
0328 nx       = si(2);

```

```

0329 nc    = si(1);
0330 nf    = length(f);
0331 Txy    = zeros(nc,nc,nf);
0332 Sig    = zeros(nc,nc,nf);
0333 ENBW    = zeros(nc,nc,nf);
0334
0335 disp_each = round(nf/100)*10;
0336
0337 for fi=1:nf
0338
0339     % compute DFT exponent and window
0340     l = L(fi);
0341     switch win.name
0342     case 'Kaiser'
0343         win = specwin(win.name, l, win.psl1);
0344     otherwise
0345         win = specwin(win.name, l);
0346     end
0347
0348     p      = 1i * twopi * m(fi)/l.*[0:l-1];
0349     C      = win.win .* exp(p);
0350     Xolap  = (1-win.rov/100);
0351
0352     if mod(fi,disp_each) == 0 || fi == 1 || fi == nf
0353         disp(sprintf('++ computing frequency %d of %d: %f Hz', fi, nf, f(fi)));
0354     end
0355
0356     % Loop over input channels
0357     for j=1:nc
0358         % loop over output channels
0359         for k=1:nc
0360
0361             % Count N segments
0362             start = 1;
0363             nsecs = 0;
0364             while start + l - 1 <= nx
0365                 % make next step
0366                 start = floor(start + l*Xolap);
0367                 nsecs = nsecs + 1;
0368             end
0369
0370             % do segments
0371             A      = 0.0;
0372             T      = zeros(nsecs,1);
0373             i2     = zeros(nsecs,1);
0374             start = 1;
0375             nsecs = 0;
0376             p      = 1;
0377             while start + l - 1 <= nx
0378
0379                 % get segments
0380                 idx = start:start+l-1;

```

```

0381         xi    = X(j,idx).';
0382         xo    = X(k,idx).';
0383
0384         % detrend segments
0385         switch order
0386             case -1
0387                 % do nothing
0388             case 0
0389                 xi = xi - mean(xi);
0390                 xo = xo - mean(xo);
0391             case 1
0392                 xi = detrend(xi);
0393                 xo = detrend(xo);
0394             otherwise
0395                 xi = polydetrend(xi, order);
0396                 xo = polydetrend(xo, order);
0397         end
0398
0399         % compute DFT
0400         ti    = C*xi;
0401         to    = C*xo;
0402         % i2(p) = ti*conj(ti);
0403         T(p)  = (to .* conj(ti)); % ./ (abs(ti).*abs(to)) ; % (to/ti);
0404         % A      = A + T(p)*i2(p);
0405         A      = A + T(p);
0406
0407         % make next step
0408         start = floor(start + l*Xolap);
0409         nsegs = nsegs + 1;
0410         p      = p + 1;
0411     end
0412
0413     A2ns      = 2.0*A/nsegs;
0414     S1         = win.ws;
0415     S12        = S1*S1;
0416     S2         = win.ws2;
0417     ENBW(j,k,fi) = fs*S2/S12;
0418
0419     sw = sum(i2);
0420
0421     Txy(j,k,fi) = 2.0*A/nsegs/fs/S2;
0422
0423     % compute variance
0424     if computeVariance
0425         A      = 0;
0426         start  = 1;
0427         p      = 1;
0428         scale  = fs*S2;
0429         while start + l - 1 <= nx
0430             % for p=1:ns
0431                 S  = (T(p)/scale-Txy(j,k,fi))^2;
0432                 A  = A + S * i2(p);

```

```
0433         % make next step
0434         start = floor(start + l*Xolap);
0435         p      = p + 1;
0436     end
0437     Sig(j,k,fi) = A/sw;
0438 end
0439
0440     end % End loop over output channels
0441 end % End loop over input channels
0442 end % End loop over frequencies
0443
0444 varargout{1} = Txy;
0445 if nargout == 2
0446     varargout{2} = ENBW;
0447 elseif nargout == 3
0448     varargout{3} = Sig;
0449     varargout{2} = ENBW;
0450 end
0451
0452
0453 % END
```

[Home](#) > [classes](#) > [@pzmodel](#) > save.m

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 1.1 2007/08/29 18:39:26 hewitson Exp \$

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

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

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE a pzmodel object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for pzmodel objects.
0008 %
0009 % CALL:          Save a pzmodel object as a .mat file.
0010 %               >> save(pzm, 'blah.mat')
0011 %               Save a pzmodel object as an XML file.
0012 %               >> save(pzm, 'blah.xml')
0013 %               Save a pzmodel object to file specified in plist.
0014 %               >> save(pzm, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:      $Id: save.m,v 1.1 2007/08/29 18:39:26 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(pzmodel, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'pzmodel') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041 ALGONAME = mfilename;
0042 VERSION  = '$Id: save.m,v 1.1 2007/08/29 18:39:26 hewitson Exp $';

```

```
0043
0044 if nargin ~= 2
0045     help mfilename;
0046     error('### incorrect inputs');
0047 end
0048
0049 % get object
0050 obj = varargin{1};
0051 if isa(varargin{2}, 'plist')
0052     pl = varargin{2};
0053 elseif ischar(varargin{2})
0054     pl = plist(param('filename', varargin{2}));
0055 else
0056     help mfilename;
0057     error('### incorrect inputs');
0058 end
0059
0060 % Save object
0061 ltpda_saveobj(obj, pl);
0062
0063 % Get default params
0064 function plo = getDefaultPL()
0065
0066 disp('* creating default plist...');
0067 plo = plist();
0068 plo = append(plo, param('filename', 'object.xml'));
0069 disp('* done. ');
0070
0071
0072 % END
```

[Home](#) > [classes](#) > [@history](#) > xmladd.m

xmladd

PURPOSE

XMLADD add an history object to the given node of the input XML DOM

SYNOPSIS

```
function xml = xmladd(hs, xml, nodename, node)
```

DESCRIPTION

XMLADD add an history object to the given node of the input XML DOM object.

M Hewitson 14-02-07

\$Id: xmladd.m,v 1.2 2007/10/15 17:16:56 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [get](#) GET get history object properties.
- [xml](#) XML convert a specwin object to an DOM XML object.
- [xmladd](#) XMLADD add an history object to the given node of the input XML DOM

This function is called by:

- [xml](#) XML convert a specwin object to an DOM XML object.
- [xmladd](#) XMLADD add an history object to the given node of the input XML DOM

SOURCE CODE

```
0001 function xml = xmladd(hs, xml, nodename, node)
0002
0003 % XMLADD add an history object to the given node of the input XML DOM
0004 % object.
0005 %
0006 % M Hewitson 14-02-07
0007 %
0008 % $Id: xmladd.m,v 1.2 2007/10/15 17:16:56 ingo Exp $
```

```

0009 %
0010
0011 el = xml.docNode.createElement(nodename);
0012
0013 for j=1:length(hs)
0014     h = hs(j);
0015     % Now add object
0016     elc = xml.docNode.createElement('history');
0017
0018     %--- add myself first
0019
0020     % add name
0021     elcc = xml.docNode.createElement('name');
0022     elcc.appendChild(xml.docNode.createTextNode(get(h, 'name')));
0023     elc.appendChild(elcc);
0024
0025     % add version
0026     elcc = xml.docNode.createElement('version');
0027     elcc.appendChild(xml.docNode.createTextNode(get(h, 'version')));
0028     elc.appendChild(elcc);
0029
0030     % add created
0031     xml = xmladd(get(h, 'created'), xml, 'created', elc);
0032
0033     % add n
0034     elcc = xml.docNode.createElement('n');
0035     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(get(h, 'n'))));
0036     elc.appendChild(elcc);
0037
0038     % add pn
0039     elcc = xml.docNode.createElement('pn');
0040     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(get(h, 'pn'))));
0041     elc.appendChild(elcc);
0042
0043     % add consver
0044     elcc = xml.docNode.createElement('consver');
0045     elcc.appendChild(xml.docNode.createTextNode(get(h, 'consver')));
0046     elc.appendChild(elcc);
0047
0048     % add invars
0049     elcc = xml.docNode.createElement('invars');
0050     invars = get(h, 'invars');
0051     istr = '';
0052     for jj=1:length(invars)
0053         istr = [istr ' ' invars{jj}];
0054     end
0055     elcc.appendChild(xml.docNode.createTextNode(istr));
0056     elc.appendChild(elcc);
0057
0058     % add plist

```

```
0059   pl = get(h, 'plist');
0060   if ~isempty(pl)
0061       xml = xmladd(pl, xml, 'plist', elc);
0062   end
0063
0064   % add inhists
0065   ih = get(h, 'inhists');
0066   if ~isempty(ih)
0067       xml = xmladd(ih, xml, 'inhists', elc);
0068   end
0069
0070   el.appendChild(elc);
0071 end
0072 node.appendChild(el);
0073
0074
0075
0076 % END
```

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 1.9 2008/01/08 16:39:12 ingo 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:

SOURCE CODE

```
0001 function txt = display(tsdatas, varargin)
0002 % DISPLAY overloads display functionality for tsdata objects.
```

```

0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY overloads display functionality for tsdata objects.
0007 %
0008 % CALL:          txt      = display(tsdata)
0009 %
0010 % INPUT:         tsdata - tsdta object
0011 %
0012 % OUTPUT:        txt      - cell array with strings to display the tsdata object
0013 %
0014 % VERSION:       $Id: display.m,v 1.9 2008/01/08 16:39:12 ingo Exp $
0015 %
0016 % HISTORY:       31-01-2007 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: display.m,v 1.9 2008/01/08 16:39:12 ingo Exp $';
0022
0023 % Check if this is a call for parameters
0024 if nargin == 2
0025     if isa(tsdatas, 'tsdata') && ischar(varargin{1})
0026         in = char(varargin{1});
0027         if strcmp(in, 'Params')
0028             txt = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             txt = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 txt = {};
0038
0039 for i=1:numel(tsdatas)
0040     ts = tsdatas(i);
0041     banner = sprintf('----- tsdata %02d -----', i);
0042     txt{end+1} = banner;
0043
0044     txt{end+1} = ' ';
0045
0046     xdata = get(ts, 'x');
0047     ydata = get(ts, 'y');
0048     xinfo = whos('xdata');
0049     yinfo = whos('ydata');
0050
0051     txt{end+1} = sprintf('  name: %s', get(ts, 'name'));
0052     txt{end+1} = sprintf('    fs: %g', get(ts, 'fs'));
0053     txt{end+1} = sprintf('    x: [%d %d], %s', xinfo.size(1), xinfo.size(2),
xinfo.class);

```

```
0054     txt{end+1} = sprintf('      y:  [%d %d], %s', yinfo.size(1), yinfo.size(2),  
yinfo.class);  
0055     txt{end+1} = sprintf('xunits: %s', get(ts, 'xunits'));  
0056     txt{end+1} = sprintf('yunits: %s', get(ts, 'yunits'));  
0057     txt{end+1} = sprintf('  nsecs: %d', get(ts, 'nsecs'));  
0058     txt{end+1} = sprintf('      t0: %s', char(get(ts, 't0')));  
0059  
0060     banner_end(1:length(banner)) = '-';  
0061     txt{end+1} = banner_end;  
0062  
0063     txt{end+1} = ' ';  
0064     txt{end+1} = ' ';  
0065 end  
0066  
0067 if nargout == 0  
0068     for ii=1:length(txt)  
0069         disp(txt{ii});  
0070     end  
0071 end  
0072  
0073 % END
```


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

```
0001 function ltpda_legendAdd(varargin)
0002 % LTPDA_LEGENDADD Add a string to the current legend.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
```

```

0006 % DESCRIPTION: LTPDA_LEGENDADD Add a string to the current legend.
0007 %
0008 % CALL:          legendAdd(fig, 'string')
0009 %                plot (curr_axes_handle, history, arg)
0010 %
0011 % INPUT:         fig      = figure handle (gcf, for example)
0012 %                string = string to add
0013 %
0014 % VERSION:       $Id: ltpda_legendAdd.m,v 1.4 2007/06/07 08:56:24 ingo Exp $
0015 %
0016 % HISTORY: 26-01-07 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%
0020
0021
0022 if nargin ~=2
0023     error('usage: legendAdd(fig, 'strin')');
0024 end
0025
0026 fig = varargin{1};
0027 strin = varargin{2};
0028
0029 if ~ischar(strin)
0030     error('arg 2 is not a string > usage: legendAdd(fig, 'strin')');
0031 end
0032 % if ~isobject(fig)
0033 %     error('arg 1 is not an object > usage: legendAdd(fig, 'strin')');
0034 % end
0035
0036 % get the current legend strings
0037
0038
0039 ch = get(gcf, 'Children');
0040 leg = [];
0041
0042 for j=1:length(ch)
0043
0044     tag = get(ch(j), 'Tag');
0045
0046     if strcmp(tag, 'legend')
0047         leg = get(ch(j));
0048     end
0049
0050 end
0051 str = [];
0052 % get(leg, 'XLabel')
0053 if ~isempty(leg)
0054     for j=length(leg.Children):-1:1
0055
0056         ch = leg.Children(j);
0057

```

```
0058     tag = get(ch, 'Tag');
0059
0060     if ~strcmp(tag, '')
0061
0062         str = strvcat(str, tag);
0063     end
0064 end
0065 end
0066
0067 str = strvcat(str, strin);
0068
0069 legend(str);
0070
```

[Home](#) > [classes](#) > [@specwin](#) > save.m

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 1.1 2007/08/29 18:39:53 hewitson Exp \$

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

```
>> pl = save(specwin, 'Params')
```

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE a specwin object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for specwin objects.
0008 %
0009 % CALL:          Save a specwin object as a .mat file.
0010 %                >> save(w, 'blah.mat')
0011 %                Save a specwin object as an XML file.
0012 %                >> save(w, 'blah.xml')
0013 %                Save a specwin object to file specified in plist.
0014 %                >> save(w, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:      $Id: save.m,v 1.1 2007/08/29 18:39:53 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(specwin, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'specwin') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041 ALGONAME = mfilename;
0042 VERSION  = '$Id: save.m,v 1.1 2007/08/29 18:39:53 hewitson Exp $';

```

```
0043
0044 if nargin ~= 2
0045     help mfilename;
0046     error('### incorrect inputs');
0047 end
0048
0049 % get object
0050 obj = varargin{1};
0051 if isa(varargin{2}, 'plist')
0052     pl = varargin{2};
0053 elseif ischar(varargin{2})
0054     pl = plist(param('filename', varargin{2}));
0055 else
0056     help mfilename;
0057     error('### incorrect inputs');
0058 end
0059
0060 % Save object
0061 ltpda_saveobj(obj, pl);
0062
0063 % Get default params
0064 function plo = getDefaultPL()
0065
0066 disp('* creating default plist...');
0067 plo = plist();
0068 plo = append(plo, param('filename', 'object.xml'));
0069 disp('* done. ');
0070
0071
0072 % END
```

[Home](#) > [classes](#) > [@mfir](#) > xmladd.m

xmladd

PURPOSE

XMLADD add an MFIR object to the given node of the input XML DOM

SYNOPSIS

function xml = xmladd(ps, xml, nodename, node)

DESCRIPTION

XMLADD add an MFIR object to the given node of the input XML DOM object.

M Hewitson 14-02-07

\$Id: xmladd.m,v 1.2 2007/11/02 13:18:29 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [get](#) GET get mfir properties.
- [xml](#) XML convert an mfir object to an DOM XML object.
- [xmladd](#) XMLADD add an MFIR object to the given node of the input XML DOM

This function is called by:

- [xml](#) XML convert an mfir object to an DOM XML object.
- [xmladd](#) XMLADD add an MFIR object to the given node of the input XML DOM

SOURCE CODE

```
0001 function xml = xmladd(ps, xml, nodename, node)
0002
0003 % XMLADD add an MFIR object to the given node of the input XML DOM
0004 % object.
0005 %
0006 % M Hewitson 14-02-07
0007 %
0008 % $Id: xmladd.m,v 1.2 2007/11/02 13:18:29 ingo Exp $
0009 %
```

```

0010 el = xml.docNode.createElement(nodename);
0011
0012 for j=1:length(ps)
0013     p = ps(j);
0014     elc = xml.docNode.createElement('mfir');
0015
0016
0017     % add name
0018     elcc = xml.docNode.createElement('name');
0019     elcc.appendChild(xml.docNode.createTextNode(get(p, 'name')));
0020     elc.appendChild(elcc);
0021
0022     % add fs
0023     elcc = xml.docNode.createElement('fs');
0024     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(get(p, 'fs'))));
0025     elc.appendChild(elcc);
0026
0027     % add ntaps
0028     elcc = xml.docNode.createElement('ntaps');
0029     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(get(p, 'ntaps'))));
0030     elc.appendChild(elcc);
0031
0032     % add a
0033     a = get(p, 'a');
0034     elcc = xml.docNode.createElement('a');
0035     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(a)));
0036     elc.appendChild(elcc);
0037
0038     % add a
0039     a = get(p, 'gd');
0040     elcc = xml.docNode.createElement('gd');
0041     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(a)));
0042     elc.appendChild(elcc);
0043
0044     % add gain
0045     elcc = xml.docNode.createElement('gain');
0046     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(get(p, 'gain'))));
0047     elc.appendChild(elcc);
0048
0049     % add histout
0050     oh = get(p, 'histout');
0051     elcc = xml.docNode.createElement('histout');
0052     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(oh)));
0053     elc.appendChild(elcc);
0054
0055     % add infile
0056     elcc = xml.docNode.createElement('infile');
0057     elcc.appendChild(xml.docNode.createTextNode(get(p, 'infile')));
0058     elc.appendChild(elcc);
0059
0060     % add plist

```



```
0061   pl = get(p, 'plist');
0062   if isa(pl, 'plist')
0063       xml = xmladd(pl, xml, 'plist', elc);
0064   end
0065
0066   % add created
0067   xml = xmladd(get(p, 'created'), xml, 'created', elc);
0068
0069   % add version
0070   elcc = xml.docNode.createElement('version');
0071   elcc.appendChild(xml.docNode.createTextNode(get(p, 'version')));
0072   elc.appendChild(elcc);
0073
0074   el.appendChild(elc);
0075 end
0076 node.appendChild(el);
0077
0078
0079
0080 % END
```

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 1.6 2008/01/08 16:39:12 ingo 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:

SOURCE CODE

```
0001 function txt = display(xydatas, varargin)
0002 % DISPLAY overloads display functionality for xydata objects.
```

```

0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY overloads display functionality for xydata objects.
0007 %
0008 % CALL:          txt      = display(xydata)
0009 %
0010 % INPUT:         xydata - tsdta object
0011 %
0012 % OUTPUT:        txt      - cell array with strings to display the xydata object
0013 %
0014 % VERSION:       $Id: display.m,v 1.6 2008/01/08 16:39:12 ingo Exp $
0015 %
0016 % HISTORY:       31-01-2007 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: display.m,v 1.6 2008/01/08 16:39:12 ingo Exp $';
0022
0023 % Check if this is a call for parameters
0024 if nargin== 2
0025     if isa(xydatas, 'xydata') && ischar(varargin{1})
0026         in = char(varargin{1});
0027         if strcmp(in, 'Params')
0028             txt = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             txt = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 txt = {};
0038
0039 for i=1:numel(xydatas)
0040     xy = xydatas(i);
0041     banner = sprintf('----- xydata %02d -----', i);
0042     txt{end+1} = banner;
0043
0044     txt{end+1} = ' ';
0045
0046     xdata = get(xy, 'x');
0047     ydata = get(xy, 'y');
0048     xinfo = whos('xdata');
0049     yinfo = whos('ydata');
0050
0051     txt{end+1} = sprintf('  name: %s', get(xy, 'name'));
0052     txt{end+1} = sprintf('      x: [%d %d], %s', xinfo.size(1), xinfo.size(2),
xinfo.class);
0053     txt{end+1} = sprintf('      y: [%d %d], %s', yinfo.size(1), yinfo.size(2),

```

```
yinfo.class);
0054   txt{end+1} = sprintf('xunits:  %s', get(xy, 'xunits'));
0055   txt{end+1} = sprintf('yunits:  %s', get(xy, 'yunits'));
0056
0057   banner_end(1:length(banner)) = '-';
0058   txt{end+1} = banner_end;
0059
0060   txt{end+1} = ' ';
0061   txt{end+1} = ' ';
0062 end
0063
0064 if nargout == 0
0065     for ii=1:length(txt)
0066         disp(txt{ii});
0067     end
0068 end
0069
0070 % END
```

[Home](#) > [m](#) > [math](#) > ltpda_lincom.m

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

```

0001 function varargout = ltpda_lincom(varargin)
0002 % LTPDA_LINCOM make the linear combination of the input analysis objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_LINCOM make the linear combination of the
0007 %               input analysis objects.
0008 %
0009 % CALL:         b = lincom(a1,a2,a3,...,pl)
0010 %               b = lincom(a1,a2,a3,...,aN)
0011 %
0012 %               If no parameter list is specified, one of the analysis objects
0013 %               should contain a cdata type with the coefficients inside.
0014 %
0015 % INPUTS:       aN - a list of analysis objects of the same type
0016 %               pl - a parameter list
0017 %
0018 % OUTPUTS:      b  - output analysis object
0019 %
0020 %               If the last input argument is a parameter list (plist) it is
0021 %               used.
0022 %               The following parameters are recognised.
0023 % PARAMETERS:   'coeffs' - vector of coefficients, same length as number of
0024 %               input AOs
0025 %               The following call returns a parameter list object that
0026 %               contains the default parameter values:
0027 %
0028 %               >> pl = ltpda_lincom('Params')
0029 %
0030 % HISTORY:      14-02-07 M Hewitson
0031 %               Creation.
0032 %
0033 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0034
0035 % Check if this is a call for parameters
0036 if nargin == 1
0037     in = char(varargin{1});

```

```

0038     if strcmp(in, 'Params')
0039         varargout{1} = getDefaultPL();
0040         return
0041     end
0042 end
0043
0044 % capture input variable names
0045 invars = {};
0046 for j=1:nargin
0047     invars = [invars cellstr(inputname(j))];
0048 end
0049
0050 ALGONAME = mfilename;
0051 VERSION  = '$Id: ltpda_lincom.m,v 1.13 2007/11/09 14:07:39 ingo Exp $';
0052
0053 % Initialise output
0054 b  = [];
0055 ho = [];
0056 %% Look at inputs
0057
0058 % check for plist and count
0059 % AOs with fsdata and tsdata
0060 pl      = [];
0061 coeffs  = [];
0062 na = 0;
0063 for j=1:nargin
0064     a = varargin{j};
0065     if isa(a, 'plist')
0066         pl = a;
0067     end
0068 end
0069 % if a cdata is supplied, use
0070 % this instead of a plist
0071 as = [];
0072 for j=1:nargin
0073     a = varargin{j};
0074     for k=1:length(a)
0075         ak = a(k);
0076         if isa(ak, 'ao')
0077             d = ak.data;
0078             % is this a constant data
0079             if isa(d, 'cdata')
0080                 coeffs = d.y;
0081                 % add history of this AO
0082                 ho = [ho ak.hist];
0083 %                 % Add coefficients to plist for history purposes
0084 %                 pl = plist(param('coeffs', coeffs));
0085             elseif isa(d, 'tsdata')
0086                 as = [as ak];
0087             end
0088         end
0089     end

```

```

0090 end
0091
0092 % If a plist is supplied and no cdata, use plist
0093 if ~isempty(pl) && isempty(coeffs)
0094     coeffs = find(pl, 'coeffs');
0095 end
0096
0097 na = length(as);
0098 nc = length(coeffs);
0099
0100 if na < 2
0101     error('### at least two analysis objects are needed to form a linear
combination.');
```

```

0102 end
0103
0104 if na ~= nc
0105     disp(sprintf('Num AOs input: %d', na));
0106     disp(sprintf('Num Coeffs input: %d', nc));
0107     error('### specify one coefficient per analysis object.');
```

```

0108 end
0109
0110 % loop over all inputs AOs
0111 nameStr = [];
0112 y = 0;
0113 for j=1:na
0114
0115     % get this data
0116     aa = as(j);
0117     da = aa.data;
0118     dainfo = whos('da');
```

```

0119     % store the first AO data type to
0120     % check against the others
0121     if j==1
0122         d = da;
0123         dinfo = whos('d');
```

```

0124     end
0125     % check we know what type of data this is
0126     if isa(da, 'tsdata') || isa(da, 'fsdata')
0127         ya = da.y;
0128     else
0129         error('### unknown data type.');
```

```

0130     end
0131
0132     % check against first AO data
0133     if ~strcmp(dainfo.class, dinfo.class)
0134         error('### all analysis objects should have the same data type.');
```

```

0135     end
0136
0137     % All time-series vectors should be the same sample rate
0138     if isa(d, 'tsdata')
0139         if da.fs ~= d.fs
0140             error('### all analysis objects should contain same sample rate data.');
```



```

0141     end
0142 end
0143
0144 % All vectors need to be the same length
0145 if y~=0
0146     if length(y) ~= length(ya)
0147         error('### all vectors should be the same length.');
```

```

0148     end
0149 end
0150 % get the appropriate coefficient and add to linear combination
0151 c = coeffs(j);
0152 y = y + c.*ya;
0153 % add name to output str
0154 if j<=length(invars)
0155     if isempty(invars{j})
0156         n = aa.name;
0157     else
0158         n = invars{j};
0159     end
0160 else
0161     n = aa.name;
0162 end
0163 nameStr = [nameStr '+' num2str(c) sprintf(' *%s', n)];
0164
0165 % add history of this AO
0166 ho = [ho aa.hist];
0167 end
0168
0169 % clean up nameStr
0170 if nameStr(1) == '+'
0171     nameStr = nameStr(2:end);
0172 end
0173
0174 % Make output analysis object
0175
0176 % create new output data
0177 if isa(d, 'tsdata')
0178     data = tsdata(d.x, y);
0179     data = set(data, 'name', nameStr);
0180 elseif isa(d, 'fsdata')
0181     data = fsdata(d.x, y, d.fs);
0182     data = set(data, 'name', nameStr);
0183 else
0184     error('### unknown data type.');
```

```

0185 end
0186
0187 % create new output history
0188 h = history(ALGONAME, VERSION, pl, ho);
0189 h = set(h, 'invars', invars);
0190
0191 % make output analysis object
0192 b = ao(data, h);

```

```
0193
0194 % set name
0195 b = set(b, 'name', nameStr);
0196
0197 varargout{1} = b;
0198
0199 %-----
0200 % Get default params
0201 function plo = getDefaultPL()
0202
0203 plo = plist();
0204
0205
0206 % END
```

[Home](#) > [classes](#) > [@time](#) > save.m

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 1.1 2007/08/29 18:40:16 hewitson Exp \$

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

```
>> pl = save(time, 'Params')
```

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE a time object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for time objects.
0008 %
0009 % CALL:          Save a time object as a .mat file.
0010 %                >> save(t, 'blah.mat')
0011 %                Save a time object as an XML file.
0012 %                >> save(t, 'blah.xml')
0013 %                Save a time object to file specified in plist.
0014 %                >> save(t, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:      $Id: save.m,v 1.1 2007/08/29 18:40:16 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(time, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'time') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041
0042 ALGONAME = mfilename;

```

```
0043 VERSION = '$Id: save.m,v 1.1 2007/08/29 18:40:16 hewitson Exp $';
0044
0045 if nargin ~= 2
0046     help mfilename;
0047     error('### incorrect inputs');
0048 end
0049
0050 % get object
0051 obj = varargin{1};
0052 if isa(varargin{2}, 'plist')
0053     pl = varargin{2};
0054 elseif ischar(varargin{2})
0055     pl = plist(param('filename', varargin{2}));
0056 else
0057     help mfilename;
0058     error('### incorrect inputs');
0059 end
0060
0061 % Save object
0062 ltpda_saveobj(obj, pl);
0063
0064 % Get default params
0065 function plo = getDefaultPL()
0066
0067 disp('* creating default plist...');
0068 plo = plist();
0069 plo = append(plo, param('filename', 'object.xml'));
0070 disp('* done. ');
0071
0072
0073 % END
```

[Home](#) > [classes](#) > [@miir](#) > xmladd.m

xmladd

PURPOSE

XMLADD add an MIIR object to the given node of the input XML DOM

SYNOPSIS

```
function xml = xmladd(ps, xml, nodename, node)
```

DESCRIPTION

XMLADD add an MIIR object to the given node of the input XML DOM object.

M Hewitson 14-02-07

\$Id: xmladd.m,v 1.2 2007/10/24 10:55:11 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [get](#) GET get miir properties.
- [xml](#) XML convert an miir object to an DOM XML object.
- [xmladd](#) XMLADD add an MIIR object to the given node of the input XML DOM

This function is called by:

- [xml](#) XML convert an miir object to an DOM XML object.
- [xmladd](#) XMLADD add an MIIR object to the given node of the input XML DOM

SOURCE CODE

```
0001 function xml = xmladd(ps, xml, nodename, node)
0002
0003 % XMLADD add an MIIR object to the given node of the input XML DOM
0004 % object.
0005 %
0006 % M Hewitson 14-02-07
0007 %
0008 % $Id: xmladd.m,v 1.2 2007/10/24 10:55:11 ingo Exp $
0009 %
```

```

0010 el = xml.docNode.createElement(nodename);
0011
0012 for j=1:length(ps)
0013     p = ps(j);
0014     elc = xml.docNode.createElement('miir');
0015
0016
0017     % add version
0018     elcc = xml.docNode.createElement('version');
0019     elcc.appendChild(xml.docNode.createTextNode(get(p, 'version')));
0020     elc.appendChild(elcc);
0021
0022     % add name
0023     elcc = xml.docNode.createElement('name');
0024     elcc.appendChild(xml.docNode.createTextNode(get(p, 'name')));
0025     elc.appendChild(elcc);
0026
0027     % add created
0028     xml = xmladd(get(p, 'created'), xml, 'created', elc);
0029
0030     % add fs
0031     elcc = xml.docNode.createElement('fs');
0032     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(get(p, 'fs'))));
0033     elc.appendChild(elcc);
0034
0035     % add ntaps
0036     elcc = xml.docNode.createElement('ntaps');
0037     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(get(p, 'ntaps'))));
0038     elc.appendChild(elcc);
0039
0040     % add gain
0041     elcc = xml.docNode.createElement('gain');
0042     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(get(p, 'gain'))));
0043     elc.appendChild(elcc);
0044
0045     % add infile
0046     elcc = xml.docNode.createElement('infile');
0047     elcc.appendChild(xml.docNode.createTextNode(get(p, 'infile')));
0048     elc.appendChild(elcc);
0049
0050     % add plist
0051     pl = get(p, 'plist');
0052     if isa(pl, 'plist')
0053         xml = xmladd(pl, xml, 'plist', elc);
0054     end
0055
0056     % add a
0057     a = get(p, 'a');
0058     elcc = xml.docNode.createElement('a');
0059     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(a)));

```

```
0060     elc.appendChild(elcc);
0061
0062     % add b
0063     b = get(p, 'b');
0064     elcc = xml.docNode.createElement('b');
0065     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(b)));
0066     elc.appendChild(elcc);
0067
0068     % add histin
0069     ih = get(p, 'histin');
0070     elcc = xml.docNode.createElement('histin');
0071     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(ih)));
0072     elc.appendChild(elcc);
0073
0074     % add histout
0075     oh = get(p, 'histout');
0076     elcc = xml.docNode.createElement('histout');
0077     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(oh)));
0078     elc.appendChild(elcc);
0079
0080     el.appendChild(elc);
0081 end
0082
0083 node.appendChild(el);
0084
0085
0086
0087 % END
```


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 1.2 2008/01/08 16:39:12 ingo Exp $

HISTORY:   24-12-2007 M Hewitson
           Creation

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

CROSS-REFERENCE INFORMATION

This function calls:

- [get](#) GET gets xyzdata properties.

This function is called by:

SOURCE CODE

```
0001 function txt = display(varargin)
0002 % DISPLAY overloads display functionality for xyzdata objects.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY overloads display functionality for xyzdata objects.
0007 %
0008 % CALL:          txt      = display(xyzdata)
0009 %
0010 % INPUT:         xyzdata - an xyz data object
0011 %
0012 % OUTPUT:        txt      - cell array with strings to display the xyzdata object
0013 %
0014 % VERSION:       $Id: display.m,v 1.2 2008/01/08 16:39:12 ingo Exp $
0015 %
0016 % HISTORY:       24-12-2007 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: display.m,v 1.2 2008/01/08 16:39:12 ingo Exp $';
0022
0023 % Check if this is a call for parameters
0024 if nargin == 2
0025     if ischar(varargin{2})
0026         in = char(varargin{2});
0027         if strcmp(in, 'Params')
0028             txt = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             txt = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 txt = {};
0038
0039 xyzdatas = varargin{1};
0040
0041 for i=1:numel(xyzdatas)
0042     xyz = xyzdatas(i);
0043     banner = sprintf('----- xyzdata %02d -----', i);
0044     txt{end+1} = banner;
0045
0046     txt{end+1} = ' ';
0047
0048     xdata = get(xyz, 'x');
0049     ydata = get(xyz, 'y');
0050     zdata = get(xyz, 'z');
0051     xinfo = whos('xdata');
0052     yinfo = whos('ydata');
0053     zinfo = whos('zdata');
0054
0055     txt{end+1} = sprintf('  name:  %s', get(xyz, '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('      z:  [%d %d], %s', zinfo.size(1), zinfo.size(2),  
zinfo.class);  
0059     txt{end+1} = sprintf('xunits: %s', get(xyz, 'xunits'));  
0060     txt{end+1} = sprintf('yunits: %s', get(xyz, 'yunits'));  
0061     txt{end+1} = sprintf('zunits: %s', get(xyz, 'zunits'));  
0062  
0063     banner_end(1:length(banner)) = '-';  
0064     txt{end+1} = banner_end;  
0065  
0066     txt{end+1} = '  ';  
0067     txt{end+1} = '  ';  
0068 end  
0069  
0070 if nargout == 0  
0071     for ii=1:length(txt)  
0072         disp(txt{ii});  
0073     end  
0074 end  
0075  
0076 % END
```

[Home](#) > [m](#) > [sigproc](#) > [frequency_domain](#) > ltpda_linedetect.m

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 - frequency 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

```

0001 function varargout = ltpda_linedetect(varargin)
0002 % LTPDA_LINEDETECT find spectral lines in the ao/fsdata objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_LINEDETECT find spectral lines in the ao/fsdata objects.
0007 %
0008 % CALL:          b = ltpda_linedetect(a, pl)
0009 %
0010 % PARAMETERS:   N          - max number of lines to return [default: 10]
0011 %               fsearch    - frequeuncy search interval [default: all]
0012 %               thresh     - a threshold to test normalised amplitude spectrum
0013 %               [default: 2]
0014 %               bw         - bandwidth over which to compute median [default: 20
0015 %               hc         - percent of outliers to exclude from median estimation
0016 %               [default: 0.8]
0017 %
0018 % VERSION:      $Id: ltpda_linedetect.m,v 1.8 2007/11/08 10:24:54 hewitson Exp $
0019 %
0020 % HISTORY:      14-05-2007 M Hewitson
0021 %               Creation
0022 %
0023 % The following call returns a parameter list object that contains the
0024 % default parameter values:
0025 %
0026 % >> pl = ltpda_linedetect('Params')
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029 %
0030 % Check if this is a call for parameters
0031 if nargin == 1
0032     in = char(varargin{1});
0033     if strcmp(in, 'Params')
0034         varargout{1} = getDefaultPL();
0035         return
0036     end
0037 end
0038
0039 % capture input variable names

```

```

0040 invars = {};
0041 for j=1:nargin
0042     if isa(varargin{j}, 'ao')
0043         invars = [invars cellstr(inputname(j))];
0044     end
0045 end
0046
0047 ALGONAME = mfilename;
0048 VERSION  = '$Id: ltpda_linedetect.m,v 1.8 2007/11/08 10:24:54 hewitson Exp $';
0049
0050 % Get inputs
0051 as      = [];
0052 ps      = [];
0053 queries = [];
0054 for j=1:nargin
0055     if isa(varargin{j}, 'ao')
0056         as = [as varargin{j}];
0057     end
0058     if isa(varargin{j}, 'plist')
0059         ps = [ps varargin{j}];
0060     end
0061 end
0062
0063 Na = length(as);
0064 if isempty(as)
0065     error('### Please input at least one A0.');
```

```

0066 end
0067
0068 % Combine plists
0069 if ~isempty(ps)
0070     pl = combine(ps, getDefaultPL());
0071 else
0072     pl = getDefaultPL();
0073 end
0074
0075 % Get parameters from plist
0076 N      = find(pl, 'N');
0077 fsearch = find(pl, 'fsearch');
0078 thresh  = find(pl, 'thresh');
0079 bw      = find(pl, 'bw');
0080 hc      = find(pl, 'hc');
0081
0082 % Loop over input A0s
0083 bo = [];
0084 for j=1:Na
0085
0086     a = as(j);
0087     d = a.data;
0088
0089     if isa(d, 'fsdata')
0090
0091         % Make noise-floor estimate

```

```

0092     nf = ltpda_nfest(a, pl);
0093
0094     % Make ratio
0095     r = a./nf;
0096
0097     % find lines
0098     lines = findLines(d.y, get(r.data, 'x'), get(r.data, 'y'), thresh, N,
fsearch);
0099
0100     % Make output data
0101     fs = fsdata([lines(:).f], [lines(:).a], d.fs);
0102
0103     else
0104         error('### I can only find lines in frequency-series AOs.');
```

```

0105     end
0106
0107     %----- Make output AO
0108
0109     % create new output history
0110     h = history(ALGONAME, VERSION, pl, a.hist);
0111     h = set(h, 'invars', invars);
0112
0113     % make output analysis object
0114     b = ao(fs, h);
0115
0116     % set name
0117     % name for this object
0118     if isempty(invars{j})
0119         n1 = a.name;
0120     else
0121         n1 = invars{j};
0122     end
0123     b = setnh(b, 'name', sprintf('lines(%s)', n1));
0124
0125     % Add to output array
0126     bo = [bo b];
0127
0128 end
0129
0130 % return output
0131 varargout{1} = bo;
0132
0133 %-----
0134 % find spectral lines
0135 function lines = findLines(ay, f, nay, thresh, N, fsearch)
0136
0137 % look for spectral lines
0138 l      = 0;
0139 pmax   = 0;
0140 pmaxi  = 0;
0141 line   = [];
0142 idx    = find( f>=fsearch(1) & f<=fsearch(2) );

```

```

0143 for j=1:length(idx)
0144     v = nay(idx(j));
0145     if v > thresh
0146         if v > pmax
0147             pmax = v;
0148             pmaxi = idx(j);
0149         end
0150     else
0151         if pmax > 0
0152             % find index when we have pmax
0153             fidx = pmaxi; %(find(nay(1:idx(j))==pmax));
0154             l = l+1;
0155             line(l).idx = fidx;
0156             line(l).f = f(fidx);
0157             line(l).a = ay(fidx);
0158         end
0159         pmax = 0;
0160     end
0161 end
0162
0163 % Select largest peaks
0164 lines = [];
0165 if ~isempty(line)
0166     [bl, lidx] = sort([line.a], 'descend');
0167     lidxs = lidx(1:min([N length(lidx)]));
0168     lines = line(lidxs);
0169     disp(sprintf('    + found %d lines.', length([lines.f])));
0170 else
0171     disp('    + found 0 lines.');
```

```

0172 end
0173
0174
0175
0176 %-----
0177 % get default parameter list
0178 function pl = getDefaultPL()
0179
0180 pl = plist();
0181 pl = append(pl, param('N', 10));
0182 pl = append(pl, param('fsearch', [0 1e10]));
0183 pl = append(pl, param('thresh', 2));
0184 pl = append(pl, param('bw', 20));
0185 pl = append(pl, param('hc', 0.8));
0186
0187
0188
0189
0190 % END

```


[Home](#) > [classes](#) > [@timeformat](#) > save.m

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 1.1 2007/08/29 18:40:52 hewitson Exp \$

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

```
>> pl = save(timeformat, 'Params')
```

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE a timeformat object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for timeformat objects.
0008 %
0009 % CALL:          Save a timeformat object as a .mat file.
0010 %                >> save(tf, 'blah.mat')
0011 %                Save a timeformat object as an XML file.
0012 %                >> save(tf, 'blah.xml')
0013 %                Save a timeformat object to file specified in plist.
0014 %                >> save(tf, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:      $Id: save.m,v 1.1 2007/08/29 18:40:52 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(timeformat, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'timeformat') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041
0042 ALGONAME = mfilename;

```

```
0043 VERSION = '$Id: save.m,v 1.1 2007/08/29 18:40:52 hewitson Exp $';
0044
0045 if nargin ~= 2
0046     help mfilename;
0047     error('### incorrect inputs');
0048 end
0049
0050 % get object
0051 obj = varargin{1};
0052 if isa(varargin{2}, 'plist')
0053     pl = varargin{2};
0054 elseif ischar(varargin{2})
0055     pl = plist(param('filename', varargin{2}));
0056 else
0057     help mfilename;
0058     error('### incorrect inputs');
0059 end
0060
0061 % Save object
0062 ltpda_saveobj(obj, pl);
0063
0064 % Get default params
0065 function plo = getDefaultPL()
0066
0067 disp('* creating default plist...');
0068 plo = plist();
0069 plo = append(plo, param('filename', 'object.xml'));
0070 disp('* done. ');
0071
0072
0073 % END
```

[Home](#) > [classes](#) > [@param](#) > xmladd.m

xmladd

PURPOSE

XMLADD add a parameter object to the given node of the input XML

SYNOPSIS

```
function xml = xmladd(ps, xml, nodename, node)
```

DESCRIPTION

XMLADD add a parameter object to the given node of the input XML DOM object.

M Hewitson 14-02-07

\$Id: xmladd.m,v 1.3 2008/01/17 19:30:41 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [xml](#) XML convert a param object to an DOM XML object.
- [xmladd](#) XMLADD add a parameter object to the given node of the input XML

This function is called by:

- [xml](#) XML convert a param object to an DOM XML object.
- [xmladd](#) XMLADD add a parameter object to the given node of the input XML

SOURCE CODE

```
0001 function xml = xmladd(ps, xml, nodename, node)
0002
0003 % XMLADD add a parameter object to the given node of the input XML
0004 % DOM object.
0005 %
0006 % M Hewitson 14-02-07
0007 %
0008 % $Id: xmladd.m,v 1.3 2008/01/17 19:30:41 hewitson Exp $
0009 %
```

```

0010
0011 el = xml.docNode.createElement(nodename);
0012
0013 for j=1:length(ps)
0014     p = ps(j);
0015     % Now add object
0016     elc = xml.docNode.createElement('param');
0017
0018     %% Add AO name
0019     elcc = xml.docNode.createElement('name');
0020     elcc.appendChild(xml.docNode.createTextNode(p.name));
0021     elc.appendChild(elcc);
0022
0023     % add Created
0024     xml = xmladd(p.created, xml, 'created', elc);
0025
0026     % add version
0027     elcc = xml.docNode.createElement('version');
0028     elcc.appendChild(xml.docNode.createTextNode(p.version));
0029     elc.appendChild(elcc);
0030
0031     % add key
0032     elcc = xml.docNode.createElement('key');
0033     elcc.appendChild(xml.docNode.createTextNode(p.key));
0034     elc.appendChild(elcc);
0035
0036     % add value: char
0037     if ischar(p.val)
0038         elcc = xml.docNode.createElement('value');
0039         elcc.appendChild(xml.docNode.createTextNode(p.val));
0040         elc.appendChild(elcc);
0041
0042     % add value: numeric
0043     elseif isnumeric(p.val)
0044         elcc = xml.docNode.createElement('value');
0045         elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(p.val)));
0046         elc.appendChild(elcc);
0047
0048     % add value: cell array of char
0049     elseif iscell(p.val)
0050         if isempty(p.val)
0051             elcc = xml.docNode.createElement('value');
0052             elcc.appendChild(xml.docNode.createTextNode(''));
0053             elc.appendChild(elcc);
0054         else
0055             if ischar(p.val{1})
0056                 elcc = xml.docNode.createElement('value');
0057                 str = 'CELL: ';
0058                 for jj=1:length(p.val)
0059                     str = [str ' ' p.val{jj}];
0060                 end

```

```
0061         elcc.appendChild(xml.docNode.createTextNode(str));
0062         elc.appendChild(elcc);
0063     else
0064         error('### The value of a parameter can only contain a cell array of
character');
0065     end
0066 end
0067 % add value: object
0068 else
0069     xml = xmladd(p.val, xml, 'value', elc);
0070 end
0071
0072 el.appendChild(elc);
0073 end
0074
0075 node.appendChild(el);
0076
0077
0078
0079 % END
```

[Home](#) > [classes](#) > [@zero](#) > display.m

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 1.4 2008/01/02 17:58:51 ingo 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:

SOURCE CODE

```
0001 function txt = display(p, varargin)
0002 % DISPLAY overloads display functionality for zero objects.
```

```

0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY overloads display functionality for zero objects.
0007 %
0008 % CALL:          txt  = display(zero)
0009 %
0010 % INPUT:         zero - zero object
0011 %
0012 % OUTPUT:        txt  - cell array with strings to display the zero object
0013 %
0014 % VERSION:       $Id: display.m,v 1.4 2008/01/02 17:58:51 ingo Exp $
0015 %
0016 % HISTORY:       30-01-2007 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: display.m,v 1.4 2008/01/02 17:58:51 ingo Exp $';
0022
0023 % Check if this is a call for parameters
0024 if nargin== 2
0025     if isa(p, 'zero') && ischar(varargin{1})
0026         in = char(varargin{1});
0027         if strcmp(in, 'Params')
0028             txt = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             txt = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037 txt = {};
0038
0039 for i=1:numel(p)
0040     banner = sprintf('---- zero %d ----', i);
0041     txt{end+1} = banner;
0042
0043     % get key and value
0044     name = get(p(i), 'name');
0045     f     = get(p(i), 'f');
0046     q     = get(p(i), 'q');
0047     ri    = get(p(i), 'ri');
0048
0049     % display
0050     if ~isempty(f)
0051         if q > 0.5
0052             txt{end+1} = [ sprintf('%s: %g Hz, Q=%g [' , name, f, q) num2str(ri(1))
0053                 ']]];
0054         else

```



```
0054     txt{end+1} = sprintf('%s: %g Hz', name, f);
0055     end
0056 end
0057
0058     banner_end(1:length(banner)) = '-';
0059     txt{end+1} = banner_end;
0060
0061 end
0062
0063 if nargout == 0
0064     for ii=1:length(txt)
0065         disp(txt{ii});
0066     end
0067 end
0068
0069
```

[Home](#) > [m](#) > [sigproc](#) > [frequency_domain](#) > ltpda_lpsd.m

ltpda_lpsd

PURPOSE

LTPDA_LPSD implement LPSD algorithm for analysis objects.

SYNOPSIS

function varargout = ltpda_lpsd(varargin)

DESCRIPTION

LTPDA_LPSD implement LPSD algorithm for analysis objects.

%%%

LTPDA_LPSD implement LPSD algorithm for analysis objects.

```
>> bs = ltpda_lpsd(as)    % returns ASD
or
>> bs = ltpda_lpsd(as, pl) % returns ASD
or
>> [a, axx] = ltpda_lpsd(as, pl) % returns AS and ASD
```

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)
Kmin - minimum number of averages (default 1)
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.
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_lpsd('Params')
```

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 1.15 2008/01/21 10:07:37 hewitson Exp \$

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [ltpda_compute_f](#) LTPDA_COMPUTE_F compute the frequency and resolution vectors for use with mlpsd.

This function is called by:

SUBFUNCTIONS

- [function varargout = mlpsd2\(varargin\)](#)
- [function varargout = mlpsd\(varargin\)](#)
- [function plo = getDefaultPL\(\)](#)

SOURCE CODE

```
0001 function varargout = ltpda_lpsd(varargin)
0002 % LTPDA_LPSD implement LPSD algorithm for analysis objects.
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % LTPDA_LPSD implement LPSD algorithm for analysis objects.
0006 %
0007 % >> bs = ltpda_lpsd(as)    % returns ASD
0008 % or
0009 % >> bs = ltpda_lpsd(as, pl)  % returns ASD
0010 % or
0011 % >> [a, axx] = ltpda_lpsd(as, pl) % returns AS and ASD
0012 %
0013 % Inputs:
0014 %     as - array of analysis objects
0015 %     pl - parameter list (see below)
```

```

0016 %
0017 %   Outputs:
0018 %       bs - array of analysis objects, one for each input
0019 %
0020 %   Parameter list:
0021 %       Kdes - desired number of averages    (default 100)
0022 %       Kmin - minimum number of averages    (default 1)
0023 %       Jdes - number of spectral frequencies to compute (default fs/2)
0024 %       win  - a specwin window object
0025 %           Only the design parameters of the window object are used; the
0026 %           window is recomputed for each DFT length inside the lpsd_core
0027 %           algorithm.
0028 %       Noverlap - desired overlap (default: taken from window)
0029 %       Order - order of detrending
0030 %           -1 - no detrending
0031 %           0 - subtract mean
0032 %           1 - subtract linear fit
0033 %           N - subtract fit of polynomial, order N
0034 %
0035 % The following call returns a parameter list object that contains the
0036 % default parameter values:
0037 %
0038 % >> pl = ltpda_lpsd('Params')
0039 %
0040 %
0041 % References:
0042 %     "Improved spectrum estimation from digitized time series
0043 %     on a logarithmic frequency axis", Michael Troebs, Gerhard Heinzel,
0044 %     Measurement 39 (2006) 120-129.
0045 %
0046 % M Hewitson 02-02-07
0047 %
0048 % $Id: ltpda_lpsd.m,v 1.15 2008/01/21 10:07:37 hewitson Exp $
0049 %
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051 %
0052 % Check if this is a call for parameters
0053 if nargin == 1
0054     in = char(varargin{1});
0055     if strcmp(in, 'Params')
0056         varargout{1} = getDefaultPL();
0057     return
0058 end
0059 end
0060
0061 % capture input variable names
0062 invars = {};
0063 as      = [];
0064 ps      = [];
0065 for j=1:nargin
0066     invars = [invars cellstr(inputname(j))];
0067     if isa(varargin{j}, 'ao')

```

```

0068     as = [as varargin{j}];
0069 end
0070 if isa(varargin{j}, 'plist')
0071     ps = [ps varargin{j}];
0072 end
0073 end
0074
0075
0076 ALGONAME = mfilename;
0077 VERSION  = '$Id: ltpda_lpsd.m,v 1.15 2008/01/21 10:07:37 hewitson Exp $';
0078
0079 % check plist
0080 if isempty(ps)
0081     pl = getDefaultPL();
0082 else
0083     pl = combine(ps, getDefaultPL);
0084 end
0085
0086 % initialise output array
0087 bs  = [];
0088 bxx = [];
0089
0090 % Go through each input AO
0091 na = length(as);
0092 for i=1:na
0093     a = as(i);
0094     d = a.data;
0095
0096     % check this is a time-series object
0097     if ~isa(d, 'tsdata')
0098         error('### lpsd requires tsdata (time-series) inputs.');
```

```

0120 Win = find(pl, 'Win');
0121 if isempty(Win)
0122     Win = specwin('Kaiser', 100, 250);
0123     pl = append(pl, param('Win', Win));
0124 end
0125 Nolap = find(pl, 'Nolap'); % desired overlap
0126 if isempty(Nolap) || Nolap == -1
0127     pl = remove(pl, 'Nolap'); % remove old value
0128     % use window function rov
0129     Nolap = Win.rov/100;
0130     disp(sprintf('! Using recommended Nolap of %d', Nolap))
0131     pl = append(pl, param('Nolap', Nolap));
0132 end
0133 Order = find(pl, 'Order'); % desired overlap
0134 if isempty(Order)
0135     Order = 0;
0136     disp(sprintf('! Using default detrending order of %d', Order))
0137     pl = append(pl, param('Order', Order));
0138 end
0139 Debug = find(pl, 'DEBUG');
0140
0141 % Get frequency vector
0142 [f,r,m,L,rr,rrr] = ltpda_compute_f(d.fs, length(d.y), Kdes, Kmin, Jdes,
Nolap, Debug);
0143
0144 % compute LPSD
0145 try
0146     if find(pl, 'M-FILE ONLY')
0147         % Using pure m-file version
0148         [Q, Qxx, ENBW] = mlpsd(d.y, f, r, m, L, d.fs, Win, Order, Nolap);
0149     else
0150         [Q, Qxx, ENBW] = mlpsd2(d.y, f, r, m, L, d.fs, Win, Order, Win.rov);
0151     end
0152 catch
0153     % Using pure m-file version
0154     [Q, Qxx, ENBW] = mlpsd(d.y, f, r, m, L, d.fs, Win, Order, Nolap);
0155 end
0156
0157 % create new output fsdata
0158 fs = fsdata(f, sqrt(Qxx), d.fs);
0159 fs = set(fs, 'name', sprintf('lpsd of %s', d.name));
0160 fs = set(fs, 'yunits', [d.yunits '\surdHz']);
0161 fs = set(fs, 'enbw', ENBW);
0162
0163 % create new output history
0164 h = history(ALGONAME, VERSION, pl, a.hist);
0165 h = set(h, 'invars', invars);
0166
0167 % make output analysis object
0168 b = ao(fs, h);
0169
0170 % set name

```

```

0171 if isempty(invars{i})
0172     n = a.name;
0173 else
0174     n = invars{i};
0175 end
0176 b = setnh(b, 'name', sprintf('lpsd(%s)', n));
0177 bs = [bs b];
0178
0179 % % add AS as well
0180 % if nargout == 2
0181 %     % create new output fsdata
0182 %     fs = fsdata(f, sqrt(Q), d.fs);
0183 %     fs = set(fs, 'name', sprintf('lpsd of %s', d.name));
0184 %     fs = set(fs, 'yunits', [d.yunits 'rms']);
0185 %     fs = set(fs, 'enbw', ENBW);
0186 %
0187 %     % create new output history
0188 %     h = history(ALGONAME, VERSION, pl, a.hist);
0189 %     h = set(h, 'invars', invars);
0190 %
0191 %     % make output analysis object
0192 %     b = ao(fs, h);
0193 %
0194 %     % set name
0195 %     if isempty(invars{i})
0196 %         n = a.name;
0197 %     else
0198 %         n = invars{i};
0199 %     end
0200 %     b = set(b, 'name', sprintf('lpsd(%s)', n));
0201 %     bxx = [bxx b];
0202 % end
0203
0204 end
0205
0206 varargout{1} = bs;
0207
0208 % include AS
0209 % if nargout > 1
0210 %     varargout{2} = bxx;
0211 % end
0212
0213
0214
0215 %-----
0216 %
0217 function varargout = mlpsd2(varargin)
0218
0219 % MLPSD2
0220 %
0221 % [S,Sxx,ENBW] = mlpsd2(x,f,r,m,L,fs,win,order,olap)
0222 %

```

```

0223 % M Hewitson 19-05-07
0224 %
0225 %
0226
0227
0228 disp('%%%%%%%% Using ltpda_dft.mex to compute core DFT ');
0229
0230 % Get inputs
0231 x      = varargin{1};
0232 f      = varargin{2};
0233 r      = varargin{3};
0234 m      = varargin{4};
0235 L      = varargin{5};
0236 fs     = varargin{6};
0237 win    = varargin{7};
0238 order  = varargin{8};
0239 olap   = varargin{9};
0240
0241 twopi   = 2.0*pi;
0242
0243 nx      = length(x);
0244 nf      = length(f);
0245 Sxx     = zeros(nf,1);
0246 S       = zeros(nf,1);
0247 ENBW    = zeros(nf,1);
0248
0249 disp_each = round(nf/100)*10;
0250
0251 for j=1:nf
0252
0253     if mod(j,disp_each) == 0 || j == 1 || j == nf
0254         disp(sprintf('++ computing frequency %d of %d: %f Hz', j, nf, f(j)));
0255     end
0256
0257     % compute DFT exponent and window
0258     l = L(j);
0259     switch win.name
0260     case 'Kaiser'
0261         win = specwin(win.name, l, win.psl1);
0262     otherwise
0263         win = specwin(win.name, l);
0264     end
0265
0266     p      = 1i * twopi * m(j)/l.*(0:l-1);
0267     C      = win.win .* exp(p);
0268
0269     % Core DFT part in C-mex file
0270     [A, nsegs] = ltpda_dft(x, l, C, olap, order);
0271
0272     if mod(j,disp_each) == 0 || j == 1 || j == nf
0273         disp(sprintf('    averaged %d segments', nsegs));
0274     end

```



```

0275
0276     A2ns      = 2.0*A/nsegs;
0277     S1         = win.ws;
0278     S12        = S1*S1;
0279     S2         = win.ws2;
0280     ENBW(j)    = fs*S2/S12;
0281     Sxx(j)     = A2ns/fs/S2;
0282     S(j)       = A2ns/S12;
0283
0284 end
0285
0286 varargout{1} = S;
0287 varargout{2} = Sxx;
0288 varargout{3} = ENBW;
0289
0290
0291 %-----
0292 %
0293 function varargout = mlpsd(varargin)
0294
0295 % MLPSD
0296 %
0297 % [S,Sxx,ENBW] = mlpsd(x,f,r,m,L,fs,win,order,olap)
0298 %
0299 % M Hewitson 19-05-07
0300 %
0301 %
0302
0303 % Get inputs
0304 x      = varargin{1};
0305 f      = varargin{2};
0306 r      = varargin{3};
0307 m      = varargin{4};
0308 L      = varargin{5};
0309 fs     = varargin{6};
0310 win    = varargin{7};
0311 order  = varargin{8};
0312 olap   = varargin{9};
0313
0314 twopi  = 2.0*pi;
0315
0316 nx     = length(x);
0317 nf     = length(f);
0318 Sxx    = zeros(nf,1);
0319 S      = zeros(nf,1);
0320 ENBW   = zeros(nf,1);
0321
0322 disp_each = round(nf/100)*10;
0323
0324 for j=1:nf
0325
0326     if mod(j,disp_each) == 0 || j == 1 || j == nf

```

```

0327     disp(sprintf('++ computing frequency %d of %d: %f Hz', j, nf, f(j)));
0328 end
0329
0330 % compute DFT exponent and window
0331 l = L(j);
0332 switch win.name
0333     case 'Kaiser'
0334         win = specwin(win.name, l, win.psl1);
0335     otherwise
0336         win = specwin(win.name, l);
0337 end
0338
0339 p      = 1i * twopi * m(j)/l.*[0:l-1];
0340 C      = win.win .* exp(p);
0341 Xolap = (1-olap);
0342 % do segments
0343 A = 0.0;
0344 start = 1;
0345 nsegs = 0;
0346 while start + l - 1 <= nx
0347
0348     % get segment
0349     xs = x(start:start+l-1);
0350
0351     % detrend segment
0352     switch order
0353         case -1
0354             % do nothing
0355         case 0
0356             xs = xs - mean(xs);
0357         case 1
0358             xs = detrend(xs);
0359         otherwise
0360             xs = polydetrend(xs, order);
0361     end
0362
0363     % make DFT
0364     a = C*xs;
0365     A = A + a*conj(a);
0366
0367     % make next step
0368     start = floor(start + l*Xolap);
0369     nsegs = nsegs + 1;
0370 end
0371
0372 if mod(j,disp_each) == 0 || j == 1 || j == nf
0373     disp(sprintf('    averaged %d segments', nsegs));
0374 end
0375
0376 A2ns      = 2.0*A/nsegs;
0377 S1        = win.ws;
0378 S12       = S1*S1;

```

```
0379     S2      = win.ws2;
0380     ENBW(j) = fs*S2/S12;
0381     Sxx(j)  = A2ns/fs/S2;
0382     S(j)    = A2ns/S12;
0383
0384 end
0385
0386 varargout{1} = S;
0387 varargout{2} = Sxx;
0388 varargout{3} = ENBW;
0389
0390
0391
0392 %-----
0393 % Get default params
0394 function plo = getDefaultPL()
0395
0396 disp('* creating default plist...');
0397 plo = plist();
0398 plo = append(plo, param('Nfft', -1));
0399 plo = append(plo, param('Win', specwin('Kaiser', 10, 100)));
0400 plo = append(plo, param('Nolap', -1));
0401 plo = append(plo, param('Order', -1));
0402 disp('* done.');
```

```
0403
0404 % END
```

[Home](#) > [classes](#) > [@timespan](#) > save.m

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 1.1 2007/08/29 18:41:27 hewitson Exp \$

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

```
>> pl = save(timespan, 'Params')
```

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE a timespan object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for timespan objects.
0008 %
0009 % CALL:          Save a timespan object as a .mat file.
0010 %                >> save(ts, 'blah.mat')
0011 %                Save a timespan object as an XML file.
0012 %                >> save(ts, 'blah.xml')
0013 %                Save a timespan object to file specified in plist.
0014 %                >> save(ts, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:      $Id: save.m,v 1.1 2007/08/29 18:41:27 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(timespan, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'timespan') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041
0042 ALGONAME = mfilename;

```

```
0043 VERSION = '$Id: save.m,v 1.1 2007/08/29 18:41:27 hewitson Exp $';
0044
0045 if nargin ~= 2
0046     help mfilename;
0047     error('### incorrect inputs');
0048 end
0049
0050 % get object
0051 obj = varargin{1};
0052 if isa(varargin{2}, 'plist')
0053     pl = varargin{2};
0054 elseif ischar(varargin{2})
0055     pl = plist(param('filename', varargin{2}));
0056 else
0057     help mfilename;
0058     error('### incorrect inputs');
0059 end
0060
0061 % Save object
0062 ltpda_saveobj(obj, pl);
0063
0064 % Get default params
0065 function plo = getDefaultPL()
0066
0067 disp('* creating default plist...');
0068 plo = plist();
0069 plo = append(plo, param('filename', 'object.xml'));
0070 disp('* done. ');
0071
0072
0073 % END
```

[Home](#) > [classes](#) > [@plist](#) > xmladd.m

xmladd

PURPOSE

XMLADD add a plist object to the given node of the input XML DOM

SYNOPSIS

```
function xml = xmladd(pls, xml, nodename, node)
```

DESCRIPTION

XMLADD add a plist object to the given node of the input XML DOM object.

M Hewitson 14-02-07

\$Id: xmladd.m,v 1.2 2007/10/15 17:16:56 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [get](#) GET get parameter list properties.
- [xml](#) XML convert a plist object to an DOM XML object.
- [xmladd](#) XMLADD add a plist object to the given node of the input XML DOM

This function is called by:

- [xml](#) XML convert a plist object to an DOM XML object.
- [xmladd](#) XMLADD add a plist object to the given node of the input XML DOM

SOURCE CODE

```
0001 function xml = xmladd(pls, xml, nodename, node)
0002
```

```
0003 % XMLADD add a plist object to the given node of the input XML DOM
0004 % object.
0005 %
0006 % M Hewitson 14-02-07
0007 %
0008 % $Id: xmladd.m,v 1.2 2007/10/15 17:16:56 ingo Exp $
0009 %
0010
0011
0012 el = xml.docNode.createElement(nodename);
0013
0014 for j=1:length(pls)
0015
0016     pl = pls(j);
0017     % Now add object
0018     elc = xml.docNode.createElement('plist');
0019
0020     %% Add AO name
0021     elcc = xml.docNode.createElement('name');
0022     elcc.appendChild(xml.docNode.createTextNode(pl.name));
0023     elc.appendChild(elcc);
0024
0025     % add version
0026     elcc = xml.docNode.createElement('version');
0027     elcc.appendChild(xml.docNode.createTextNode(get(pl, 'version')));
0028     elc.appendChild(elcc);
0029
0030     % add Created
0031     xml = xmladd(pl.created, xml, 'created', elc);
0032
0033     ps = get(pl, 'params');
0034     if ~isempty(ps)
0035         xml = xmladd(ps, xml, 'params', elc);
0036     end
0037     el.appendChild(elc);
0038 end
0039 node.appendChild(el);
0040
0041
0042
0043
0044
0045
0046
```


0047

0048 % END

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [analyses](#) > [stuttgart](#) > dkalman_matrices.m

dkalman_matrices

PURPOSE

%%
 %%%%%%%%%

SYNOPSIS

This is a script file.

DESCRIPTION

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
produce discrete input matrices for ltpda_dkalman function
(example)

veronika 05-07-07
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [test_dkalman](#) %%%

SOURCE CODE

```
0001 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0002 % produce discrete input matrices for ltpda_dkalman function
0003 % (example)
0004 %
0005 % veronika 05-07-07
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007
0008 om2_quad = -2.24*(10^(-6));
0009 deltaom = -7*(10^(-7));
0010
0011 t = 1/fs;
0012
0013 A = [0 1 0; -om2_quad 0 -1; 0 0 0];
0014 PHI= expm(A);
```

```

0015
0016 om2 = sqrt(2.24*(10^(-6)));
0017 Bd = [(1/(om2^2))*(cosh(om2*t)-1)*(deltaom)      (1/(om2^2))*(cosh(om2*t)-1);
0018         (1/om2)*sinh(om2*t)*(deltaom)            (1/om2)*sinh(om2*t);
0019         0      0];
0020
0021 H = [1 0 0];      %measurement matrix
0022
0023 x0 = [0  0  0]';
0024
0025 P0 = [100  0  0;
0026       0  100  0;
0027       0  0  100];
0028
0029 %calculate the Q and R matrix
0030 om2 = sqrt(2.24*(10^(-6)));
0031
0032 sigma1_quad = 0;
0033 sigma2_quad = 0;
0034 sigma3_quad = 10^(-13);
0035 sigmay_quad = 10^(-10);
0036
0037 R = sigmay_quad;
0038
0039 alpha1 = (sigma1_quad/2) -(sigma2_quad/(2*(om2^2))) +(3*sigma3_quad/(2*
(om2^4)));
0040 alpha2 = (sigma1_quad/4) +(sigma2_quad/(4*(om2^2))) +(sigma3_quad/(4*(om2^4)));
0041 beta = (1/4)*(sigma1_quad*om2 +sigma2_quad/om2 + sigma3_quad/(om2^3));
0042 epsi1 = sigma2_quad/2 -(sigma1_quad*(om2^2))/2 - sigma3_quad/(2*(om2^2));
0043 epsi2 = (sigma1_quad*(om2^2))/4 + sigma2_quad/4 + sigma3_quad/(4*(om2^2));
0044
0045 Q11 = alpha1*t +(exp(-om2*t)-exp(om2*t))*(sigma3_quad/(om2^5)) +(exp(2*om2*t)-
exp(-2*om2*t))*(alpha2/(2*om2));
0046
0047 Q12 = -beta/om2 +sigma3_quad/(om2^4) +(beta/(2*om2))*(exp(2*om2*t)+exp(-
2*om2*t)) -(sigma3_quad/(2*(om2^4)))*(exp(-om2*t)+exp(om2*t));
0048
0049 Q13 = -(sigma3_quad/(om2^2))*t+(sigma3_quad/(2*(om2^3)))*(-exp(-om2*t)+exp
(om2*t));
0050
0051 Q21 = Q12;
0052
0053 Q22 = epsi1*t+(epsi2/(2*om2))*(exp(2*om2*t)-exp(-2*om2*t));
0054
0055 Q23 = -sigma3_quad/(om2^2) +(sigma3_quad/(2*(om2^2)))*(exp(-om2*t)+exp(om2*t));
0056
0057 Q31 = Q13;
0058
0059 Q32 = Q23;
0060
0061 Q33 = sigma3_quad*t;
0062

```

```
0063 Q = [Q11  Q12  Q13;  
0064       Q21  Q22  Q23;  
0065       Q31  Q32  Q33];  
0066
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [sigproc](#) > [frequency_domain](#) > ltpda_ltfe.m

ltpda_ltfe

PURPOSE

LTPDA_LTFE implement weighted transfer-function estimation computed on a log frequency axis.

SYNOPSIS

function varargout = ltpda_ltfe(varargin)

DESCRIPTION

LTPDA_LTFE implement weighted transfer-function estimation computed on a log frequency axis.

%%%

LTPDA_LTFE implement weighted transfer-function estimation computed on a log frequency axis.

Usage:

```
>> tfs = ltpda_ltfe(as)
>> tfs = ltpda_ltfe(as, pl)
>> [tfs, sig2] = ltpda_ltfe(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]
Kmin - minimum number of averages [default: 10]
Jdes - number of spectral frequencies to compute [default: fs/4]
Win - a specwin window object
Only the design parameters of the window object are used; the window is recomputed for each DFT length inside the lpsd_core algorithm. [default: Kaiser with -200dB PSL]L]
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')
```

M Hewitson 02-02-07

\$Id: ltpda_ltfe.m,v 1.12 2007/11/08 10:24:54 hewitson Exp \$

%%%

CROSS-REFERENCE INFORMATION [^]

This function calls:

- [ltpda_compute_f](#) LTPDA_COMPUTE_F compute the frequency and resolution vectors for use with mlpsd.

This function is called by:

SUBFUNCTIONS [^]

- [function lmin = findShortestVector\(as\)](#)
- [function fs = findFsMax\(as\)](#)
- [function plo = getDefaultPL\(\)](#)
- [function varargout = mltfe\(varargin\)](#)

SOURCE CODE [^]

```
0001 function varargout = ltpda_ltfe(varargin)
0002 % LTPDA_LTFE implement weighted transfer-function estimation computed on a log
0003 % frequency axis.
0004 %
0005 % LTPDA_LTFE implement weighted transfer-function estimation computed on a
0006 % log frequency axis.
0007 %
0008 % Usage:
0009 %     >> tfs = ltpda_ltfe(as)
0010 %     >> tfs = ltpda_ltfe(as, pl)
0011 %     >> [tfs, sig2] = ltpda_ltfe(as, pl) % returns variance estimates
0012 %
0013 % Inputs:
0014 %     as - array of analysis objects
0015 %     pl - parameter list (see below)
0016 %
0017 % Outputs:
0018 %     bs - array of analysis objects, one for each input
0019 %
```

```

0020 % Parameter list:
0021 % Kdes - desired number of averages [default: 100]
0022 % Kmin - minimum number of averages [default: 10]
0023 % Jdes - number of spectral frequencies to compute [default: fs/4]
0024 % Win - a specwin window object
0025 % Only the design parameters of the window object are used; the
0026 % window is recomputed for each DFT length inside the lpsd_core
0027 % algorithm. [default: Kaiser with -200dB PSLL]
0028 % Nolap - desired overlap [default: taken from window]
0029 % Order - order of detrending.
0030 % -1 - no detrending
0031 % 0 - subtract mean
0032 % 1 - subtract linear fit
0033 % N - subtract fit of polynomial, order N
0034 %
0035 % The following call returns a parameter list object that contains the
0036 % default parameter values:
0037 %
0038 % >> pl = ltpda_ltfe('Params')
0039 %
0040 %
0041 % M Hewitson 02-02-07
0042 %
0043 % $Id: ltpda_ltfe.m,v 1.12 2007/11/08 10:24:54 hewitson Exp $
0044 %
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046 %
0047 % Check if this is a call for parameters
0048 if nargin == 1
0049     in = char(varargin{1});
0050     if strcmp(in, 'Params')
0051         varargout{1} = getDefaultPL();
0052         return
0053     end
0054 end
0055
0056 % capture input variable names
0057 invars = {};
0058 as = [];
0059 ps = [];
0060 for j=1:nargin
0061     invars = [invars cellstr(inputname(j))];
0062     if isa(varargin{j}, 'ao')
0063         as = [as varargin{j}];
0064     end
0065     if isa(varargin{j}, 'plist')
0066         ps = [ps varargin{j}];
0067     end
0068 end
0069
0070 ALGONAME = mfilename;
0071 VERSION = '$Id: ltpda_ltfe.m,v 1.12 2007/11/08 10:24:54 hewitson Exp $';

```

```

0072
0073 % initialise output array
0074 bs = [];
0075
0076 % check plist
0077 if isempty(ps)
0078     pl = getDefaultPL();
0079 else
0080     pl = combine(ps, getDefaultPL());
0081 end
0082
0083 % Go through each input AO and build a matrix of signals
0084 na    = length(as);
0085
0086 %----- Resample all AOs
0087
0088 disp('*** Resampling AOs...');
0089 fsmax = findFsMax(as);
0090 fspl  = plist(param('fsout', fsmax));
0091 bs = [];
0092 for i=1:na
0093     a = as(i);
0094     % check this is a time-series object
0095     if ~isa(a.data, 'tsdata')
0096         error('### ltpda_ltfe requires tsdata (time-series) inputs.');
```

```

0097     end
0098     % Check Fs
0099     if a.data.fs ~= fsmax
0100         warning(sprintf('!!! Resampling AO %s/%s to %f Hz', a.name, a.data.name,
fsmax));
0101         a = resample(a, fspl);
0102     end
0103     bs = [bs a];
0104 end
0105 as = bs;
0106 clear bs;
0107 disp('*** Done.');
```

```

0108
0109 %----- Truncate all vectors
0110
0111 % Get shortest vector
0112 disp('*** Truncating all vectors...');
```

```

0113 lmin = findShortestVector(as);
0114 nsecs = lmin / fsmax;
0115 bs = [];
0116 for i=1:na
0117     a = as(i);
0118     if len(a) ~= lmin
0119         warning(sprintf('!!! Truncating AO %s/%s to %d secs', a.name, a.data.name,
nsecs));
0120         bs = [bs select(a, 1:lmin)];
0121     else
```



```

0122     bs = [bs a];
0123     end
0124 end
0125 as = bs;
0126 clear bs;
0127 disp('*** Done. ');
0128
0129
0130 %----- Build signal Matrix
0131 N      = len(as(1)); % length of first signal
0132 iS     = zeros(na, N);
0133 for j=1:na
0134     a = as(j);
0135     iS(j,:) = a.data.y;
0136 end
0137
0138 %----- check input parameters
0139
0140 % Averaging
0141 Kdes = find(pl, 'Kdes'); % desired averages
0142 Kmin = find(pl, 'Kmin'); % minimum averages
0143 Jdes = find(pl, 'Jdes'); % num desired spectral frequencies
0144 if Jdes == -1
0145     Jdes = round(fsmax/4);
0146     disp(sprintf('! Using default Jdes of %d', Jdes))
0147     pl = pset(pl, 'Jdes', Jdes);
0148 end
0149 Win  = find(pl, 'Win'); % Window object
0150 Nlap = find(pl, 'Nlap'); % desired overlap
0151 if Nlap == -1
0152     % use window function rov
0153     Nlap = Win.rov/100;
0154     disp(sprintf('! Using recommended overlap of %d', Nlap))
0155     pl = pset(pl, 'Nlap', Nlap);
0156 end
0157 Order = find(pl, 'Order'); % desired overlap
0158 if isempty(Order)
0159     Order = 0;
0160     disp(sprintf('! Using default detrending order of %d', Order))
0161     pl = append(pl, param('Order', Order));
0162 end
0163
0164
0165 %----- Get frequency vector
0166 [f,r,m,L] = ltpda_compute_f(fsmax, lmin, Kdes, Kmin, Jdes, Nlap);
0167
0168 %----- compute TF Estimates
0169
0170 if nargout == 2
0171     disp('** Computing TF matrix with variance estimates')
0172     tic
0173     [Txy, ENBW, Sig] = mltfe(iS, f, r, m, L, fsmax, Win, Order);

```

```

0174     toc
0175 else
0176     disp('** Computing TF matrix')
0177     tic
0178     [Txy, ENBW] = mltfe(iS, f, r, m, L, fsmax, Win, Order);
0179     toc
0180 end
0181
0182 %----- Build output Matrix of AOs
0183 disp('** Building output AO matrix')
0184 bs = cell(na);
0185 sig = cell(na);
0186 for j=1:na % Loop over input signal
0187     bi = as(j);
0188     for k=1:na % Loop over output signal
0189         bo = as(k);
0190         % create new output fsdata
0191         fs = fsdata(f, squeeze(Txy(j,k,:)), fsmax);
0192         fs = set(fs, 'name', sprintf('LTFe(%s->%s)', bi.data.name, bo.data.name));
0193         fs = set(fs, 'yunits', [bo.data.yunits '/' bi.data.yunits]);
0194         fs = set(fs, 'enbw', squeeze(ENBW(j,k,:)));
0195         % create new output history
0196         h = history(ALGONAME, VERSION, pl, [bi.hist bo.hist]);
0197         try
0198             h = set(h, 'invars', invars([j k]));
0199         catch
0200             warning('!!! Unable to set input variable names for history. Perhaps you
input a vector of AOs.');
```

```

0201         end
0202         % make output analysis object
0203         b = ao(fs, h);
0204         % set name
0205         try
0206             if isempty(invars{k})
0207                 nk = bo.name;
0208             else
0209                 nk = invars{k};
0210             end
0211             if isempty(invars{j})
0212                 nj = bi.name;
0213             else
0214                 nj = invars{j};
0215             end
0216         catch
0217             warning('!!! Unable to set input variable names for history. Perhaps you
input a vector of AOs.');
```

```

0218             nk = bo.name;
0219             nj = bi.name;
0220         end
0221
0222         b = setnh(b, 'name', sprintf('LTFe(%s->%s)', nj, nk));
0223         % add to outputs

```

```

0224     bs(j,k) = {b};
0225
0226     % Make variance outputs
0227     if nargin == 2
0228         % create new output fsdata
0229         fs = fsdata(f, squeeze(Sig(j,k,:)), fsmax);
0230         fs = set(fs, 'name', ['\sigma^2' sprintf('(%s->%s)', bi.data.name, bo.
data.name)]);
0231         fs = set(fs, 'yunits', [bo.data.yunits '/' bi.data.yunits]);
0232         fs = set(fs, 'enbw', squeeze(ENBW(j,k,:)));
0233         % create new output history
0234         h = history(ALGONAME, VERSION, pl, [bi.hist bo.hist]);
0235         try
0236             h = set(h, 'invars', invars([j k]));
0237         catch
0238             warning('!!! Unable to set input variable names for history. Perhaps
you input a vector of AOs.');
```

```

0239         end
0240         % make output analysis object
0241         b = ao(fs, h);
0242         % set name
0243         try
0244             if isempty(invars{k})
0245                 nk = bo.name;
0246             else
0247                 nk = invars{k};
0248             end
0249             if isempty(invars{j})
0250                 nj = bi.name;
0251             else
0252                 nj = invars{j};
0253             end
0254         catch
0255             nk = bo.name;
0256             nj = bi.name;
0257         end
0258         b = setnh(b, 'name', ['\sigma^2' sprintf('(%s->%s)', nj, nk)]);
0259         % add to outputs
0260         sig(j,k) = {b};
0261
0262     end
0263 end % End loop over output signal
0264 end % End loop over input signal
0265
0266 % now we have a cell matrix of AOs but
0267 % we want a normal matrix
0268 b = [bs{:}];
0269 bs = reshape(b, na, na);
0270 varargout{1} = bs;
0271 if nargin == 2
0272     b = [sig{:}];
0273     s = reshape(b, na, na);

```

```

0274     varargout{2} = s;
0275 end
0276 %-----
0277 % Returns the length of the shortest vector in samples
0278 function lmin = findShortestVector(as)
0279
0280 lmin = 1e20;
0281 for j=1:length(as)
0282     if len(as(j)) < lmin
0283         lmin = len(as(j));
0284     end
0285 end
0286
0287
0288 %-----
0289 % Returns the max Fs of a set of AOs
0290 function fs = findFsMax(as)
0291
0292 fs = 0;
0293 for j=1:length(as)
0294     a = as(j);
0295     if a.data.fs > fs
0296         fs = a.data.fs;
0297     end
0298 end
0299
0300 %-----
0301 % Get default params
0302 function plo = getDefaultPL()
0303
0304 disp('* creating default plist...');
0305 plo = plist();
0306 plo = append(plo, param('Kdes', 100));
0307 plo = append(plo, param('Jdes', 1000));
0308 plo = append(plo, param('Kmin', 2));
0309 plo = append(plo, param('Win', specwin('Kaiser', 1, 150)));
0310 plo = append(plo, param('Nolap', -1));
0311 plo = append(plo, param('Order', 0));
0312 disp('* done.');
```

```

0313
0314
0315 %-----
0316 % Compute all TF combinations
0317 function varargout = mltfe(varargin)
0318
0319 % MLTFE
0320 %
0321 % Txy = mltfe(X,f,r,m,L,fs,win)
0322 %
0323 % M Hewitson 19-05-07
0324 %
0325 %
```

```

0326
0327 % Get inputs
0328 X      = varargin{1};
0329 f      = varargin{2};
0330 r      = varargin{3};
0331 m      = varargin{4};
0332 L      = varargin{5};
0333 fs     = varargin{6};
0334 win    = varargin{7};
0335 order  = varargin{8};
0336
0337 % do we need to compute variance?
0338 if nargout == 2
0339     computeVariance = 1;
0340 else
0341     computeVariance = 0;
0342 end
0343
0344 twopi    = 2.0*pi;
0345 twopiofs = 2.0*pi/(1.0*fs);
0346
0347 si      = size(X);
0348 nx      = si(2);
0349 nc      = si(1);
0350 nf      = length(f);
0351 Txy     = zeros(nc,nc,nf);
0352 Sig     = zeros(nc,nc,nf);
0353 ENBW    = zeros(nc,nc,nf);
0354
0355 disp_each = round(nf/100)*10;
0356
0357 for fi=1:nf
0358
0359     % compute DFT exponent and window
0360     l = L(fi);
0361     switch win.name
0362     case 'Kaiser'
0363         win = specwin(win.name, l, win.psl1);
0364     otherwise
0365         win = specwin(win.name, l);
0366     end
0367
0368     p      = 1i * twopi * m(fi)/l.*[0:l-1];
0369     C      = win.win .* exp(p);
0370     Xolap  = (1-win.rov/100);
0371
0372     if mod(fi,disp_each) == 0 || fi == 1 || fi == nf
0373         disp(sprintf('++ computing frequency %d of %d: %f Hz', fi, nf, f(fi)));
0374     end
0375
0376     % Loop over input channels
0377     for j=1:nc

```

```

0378 % loop over output channels
0379 for k=1:nc
0380
0381 % Count N segments
0382 start = 1;
0383 nsegs = 0;
0384 while start + l - 1 <= nx
0385     % make next step
0386     start = floor(start + l*Xolap);
0387     nsegs = nsegs + 1;
0388 end
0389
0390 % do segments
0391 A = 0.0;
0392 T = zeros(nsegs,1);
0393 i2 = zeros(nsegs,1);
0394
0395 start = 1;
0396 nsegs = 0;
0397 p = 1;
0398 while start + l - 1 <= nx
0399
0400     % get segments
0401     idx = start:start+l-1;
0402     xi = X(j,idx).';
0403     xo = X(k,idx).';
0404
0405     % detrend segments
0406     switch order
0407     case -1
0408         % do nothing
0409     case 0
0410         xi = xi - mean(xi);
0411         xo = xo - mean(xo);
0412     case 1
0413         xi = detrend(xi);
0414         xo = detrend(xo);
0415     otherwise
0416         xi = polydetrend(xi, order);
0417         xo = polydetrend(xo, order);
0418     end
0419
0420     % compute DFT
0421     ti = C*xi;
0422     to = C*xo;
0423     i2(p) = (ti*conj(ti));
0424     T(p) = to/ti;
0425     A = A + T(p)*i2(p);
0426
0427     % make next step
0428     start = floor(start + l*Xolap);
0429     nsegs = nsegs + 1;

```

```

0430     p      = p + 1;
0431 end
0432
0433     sw = sum(i2);
0434     Txy(j,k,fi) = conj(A/sw);
0435
0436     S1      = win.ws;
0437     S12     = S1*S1;
0438     S2      = win.ws2;
0439     ENBW(j,k,fi) = fs*S2/S12;
0440
0441     % compute variance
0442     if computeVariance
0443         A      = 0;
0444         start = 1;
0445         p      = 1;
0446         while start + l - 1 <= nx
0447 %             for p=1:ns
0448             S  = (T(p)-Txy(j,k,fi))^2;
0449             A  = A + S * i2(p);
0450             % make next step
0451             start = floor(start + l*Xolap);
0452             p      = p + 1;
0453         end
0454         Sig(j,k,fi) = abs(A/sw);
0455     end
0456
0457     end % End loop over output channels
0458 end % End loop over input channels
0459 end % End loop over frequencies
0460
0461 varargout{1} = Txy;
0462 if nargin >= 2
0463     varargout{2} = ENBW;
0464 end
0465 if nargin == 3
0466     varargout{3} = Sig;
0467 end
0468
0469 % END

```

[Home](#) > [classes](#) > [@tsdata](#) > save.m

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 1.1 2007/08/29 18:41:53 hewitson Exp \$

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

```
>> pl = save(tsdata, 'Params')
```

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE a tsdata object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for tsdata objects.
0008 %
0009 % CALL:          Save a tsdata object as a .mat file.
0010 %                >> save(d, 'blah.mat')
0011 %                Save a tsdata object as an XML file.
0012 %                >> save(d, 'blah.xml')
0013 %                Save a tsdata object to file specified in plist.
0014 %                >> save(d, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:      $Id: save.m,v 1.1 2007/08/29 18:41:53 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(tsdata, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'tsdata') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041 ALGONAME = mfilename;
0042 VERSION  = '$Id: save.m,v 1.1 2007/08/29 18:41:53 hewitson Exp $';

```

```
0043
0044 if nargin ~= 2
0045     help mfilename;
0046     error('### incorrect inputs');
0047 end
0048
0049 % get object
0050 obj = varargin{1};
0051 if isa(varargin{2}, 'plist')
0052     pl = varargin{2};
0053 elseif ischar(varargin{2})
0054     pl = plist(param('filename', varargin{2}));
0055 else
0056     help mfilename;
0057     error('### incorrect inputs');
0058 end
0059
0060 % Save object
0061 ltpda_saveobj(obj, pl);
0062
0063 % Get default params
0064 function plo = getDefaultPL()
0065
0066 disp('* creating default plist...');
0067 plo = plist();
0068 plo = append(plo, param('filename', 'object.xml'));
0069 disp('* done. ');
0070
0071
0072 % END
```

[Home](#) > [classes](#) > [@pole](#) > xmladd.m

xmladd

PURPOSE

XMLADD add a pole object to the given node of the input XML DOM

SYNOPSIS

function xml = xmladd(ps, xml, nodename, node)

DESCRIPTION

XMLADD add a pole object to the given node of the input XML DOM object.

M Hewitson 14-02-07

\$Id: xmladd.m,v 1.2 2007/10/16 17:37:21 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [get](#) GET get pole properties.
- [xml](#) XML convert a pole object to an DOM XML object.
- [xmladd](#) XMLADD add a pole object to the given node of the input XML DOM

This function is called by:

- [xml](#) XML convert a pole object to an DOM XML object.
- [xmladd](#) XMLADD add a pole object to the given node of the input XML DOM

SOURCE CODE

```
0001 function xml = xmladd(ps, xml, nodename, node)
0002
0003 % XMLADD add a pole object to the given node of the input XML DOM
0004 % object.
0005 %
0006 % M Hewitson 14-02-07
0007 %
```

```

0008 % $Id: xmladd.m,v 1.2 2007/10/16 17:37:21 ingo Exp $
0009 %
0010
0011 el = xml.docNode.createElement(nodename);
0012
0013 %     p.name     = 'real pole';
0014 %     p.f       = varargin{1};
0015 %     p.q       = -1;
0016 %     p.ri      = p.f;
0017 %     p.version = VERSION;
0018
0019 for j=1:length(ps)
0020     p = ps(j);
0021
0022     % Add pole object
0023     elc = xml.docNode.createElement('pole');
0024
0025     % add name
0026     elcc = xml.docNode.createElement('name');
0027     elcc.appendChild(xml.docNode.createTextNode(get(p, 'name')));
0028     elc.appendChild(elcc);
0029
0030     % add Created
0031     xml = xmladd(p.created, xml, 'created', elc);
0032
0033     % add version
0034     elcc = xml.docNode.createElement('version');
0035     elcc.appendChild(xml.docNode.createTextNode(get(p, 'version')));
0036     elc.appendChild(elcc);
0037
0038     % add f
0039     elcc = xml.docNode.createElement('f');
0040     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(get(p, 'f'))));
0041     elc.appendChild(elcc);
0042
0043     % add Q
0044     elcc = xml.docNode.createElement('q');
0045     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(get(p, 'q'))));
0046     elc.appendChild(elcc);
0047
0048     el.appendChild(elc);
0049 end
0050
0051 node.appendChild(el);
0052
0053

```


[Home](#) > [m](#) > [helper](#) > dunzip.m

dunzip

PURPOSE [^](#)

DUNZIP - decompress DZIP output to recover original data

SYNOPSIS [^](#)

function M = dunzip(Z)

DESCRIPTION [^](#)

DUNZIP - decompress DZIP output to recover original data

USAGE:

M = dzip(Z)

VARIABLES:

Z = compressed variable to decompress

M = decompressed output

NOTES: (1) The input variable Z is created by the DZIP function and is a vector of type uint8
(2) The decompressed output will have the same data type and dimensions as the original data provided to DZIP.
(3) See DZIP for other notes.
(4) Carefully tested, but no warranty; use at your own risk.
(5) Michael Kleder, Nov 2005

CROSS-REFERENCE INFORMATION [^](#)

This function calls:

This function is called by:

SOURCE CODE [^](#)

```
0001 function M = dunzip(Z)
```

```

0002 % DUNZIP - decompress DZIP output to recover original data
0003 %
0004 % USAGE:
0005 % M = dzip(Z)
0006 %
0007 % VARIABLES:
0008 % Z = compressed variable to decompress
0009 % M = decompressed output
0010 %
0011 % NOTES: (1) The input variable Z is created by the DZIP function and
0012 %           is a vector of type uint8
0013 %           (2) The decompressed output will have the same data type and
0014 %           dimensions as the original data provided to DZIP.
0015 %           (3) See DZIP for other notes.
0016 %           (4) Carefully tested, but no warranty; use at your own risk.
0017 %           (5) Michael Kleder, Nov 2005
0018
0019 import com.mathworks.mlwidgets.io.InterruptibleStreamCopier
0020 a=java.io.ByteArrayInputStream(Z);
0021 b=java.util.zip.InflaterInputStream(a);
0022 isc = InterruptibleStreamCopier.getInterruptibleStreamCopier;
0023 c = java.io.ByteArrayOutputStream;
0024 isc.copyStream(b,c);
0025 Q=typecast(c.toByteArray,'uint8');
0026 cn = double(Q(1)); % class
0027 nd = double(Q(2)); % # dims
0028 s = typecast(Q(3:8*nd+2),'double')'; % size
0029 Q=Q(8*nd+3:end);
0030 if cn == 3
0031     M = logical(Q);
0032 elseif cn == 4
0033     M = char(Q);
0034 else
0035     ct = {'double','single','logical','char','int8','uint8',...
0036           'int16','uint16','int32','uint32','int64','uint64'};
0037     M = typecast(Q,ct{cn});
0038 end
0039 M=reshape(M,s);
0040 return

```

ltpda_lxspecc

PURPOSE

LTPDA_LXSPEC performs cross-spectral analysis of various forms.

SYNOPSIS

function varargout = ltpda_lxspecc(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_lxspecc(X, f, r, m, L, fs, win, order, method);
[Y,Ys] = ltpda_lxspecc(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 estimates for Y

VERSION: \$Id: ltpda_xspecc.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

```

0001 function varargout = ltpda_lxspecc(varargin)
0002 % LTPDA_LXSPEC performs cross-spectral analysis of various forms.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_LXSPEC performs cross-spectral analysis of various forms.
0007 %             The function is a helper function for various higher level
0008 %             functions. It is meant to be called from other functions
0009 %             (e.g., ltpda_ltfec).
0010 %
0011 % CALL:          Y = ltpda_lxspecc(X, f, r, m, L, fs, win, order, method);
0012 %             [Y,Ys] = ltpda_lxspecc(X, f, r, m, L, fs, win, order, method);
0013 %
0014 % INPUTS:      X          - a matrix of input data samples
0015 %             f,r,m,L    - outputs of ltpda_compute_fc
0016 %             fs         - sample rate of data vectors
0017 %             win        - specwin object used to window the data
0018 %             order      - order of detrending:
0019 %                         -1 - no detrending
0020 %                         0 - subtract mean
0021 %                         1 - subtract linear fit
0022 %                         N - subtract fit of polynomial, order N
0023 %             method     - A collection of 'Xpsd', 'TFE', 'Coherc'
0024 %
0025 % OUTPUTS:     Y         - A cell array containing one NxN matrix of vectors for each
0026 %                         of the requested estimate types
0027 %             Ys        - A cell array of NxN matrices of variance estimates for Y
0028 %
0029 % VERSION:     $Id: ltpda_xspecc.m,v 1.7 2007/10/08 11:28:29 hewitson Exp $
0030 %
0031 % HISTORY:     28-10-2007 M Hewitson
0032 %             Creation
0033 %
0034 %
0035 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0036
0037 % Get inputs
0038 X      = varargin{1};
0039 f      = varargin{2};
0040 r      = varargin{3};
0041 m      = varargin{4};
0042 L      = varargin{5};

```

```

0043 fs      = varargin{6};
0044 win      = varargin{7};
0045 order    = varargin{8};
0046
0047 % do we need to compute variance?
0048 if nargin == 2
0049     computeVariance = 1;
0050 else
0051     computeVariance = 0;
0052 end
0053
0054 twopi     = 2.0*pi;
0055
0056 si        = size(X);
0057 nx        = si(1);
0058 nc        = si(2);
0059 nf        = length(f);
0060 Txy       = zeros(nc,nc,nf);
0061 Sig       = zeros(nc,nc,nf);
0062 ENBW      = zeros(nc,nc,nf);
0063
0064 switch order
0065     case -1
0066         detrender = '';
0067     case 0
0068         detrender = 'xi = xi - mean(xi);xo = xo - mean(xo)';
0069     case 1
0070         detrender = 'xi = detrend(xi);xo = detrend(xo)';
0071     otherwise
0072         detrender = 'xi = polydetrend(xi, order); xo = polydetrend(xo, order)';
0073 end
0074
0075 disp_each = round(nf/100)*10;
0076 for fi=1:nf
0077
0078     % compute DFT exponent and window
0079     l = L(fi);
0080     switch win.name
0081         case 'Kaiser'
0082             win = specwin(win.name, l, win.psl1);
0083         otherwise
0084             win = specwin(win.name, l);
0085     end
0086
0087     p      = 1i * twopi * m(fi)/l.*[0:l-1];
0088     C      = win.win .* exp(p);
0089     Xolap  = (1-win.rov/100);
0090     if mod(fi,disp_each) == 0 || fi == 1 || fi == nf
0091         disp(sprintf('++ computing frequency %03d of %03d: %f Hz', fi, nf, f(fi)));
0092     end
0093
0094     %----- Count segments

```

```

0095     nsecs = 0;
0096     start = 1;
0097     lm1    = l-1;
0098     while start + l - 1 <= nx
0099         start = floor(start + l*Xolap);
0100         nsecs = nsecs + 1;
0101     end
0102     % now allocate indices for segments
0103 %     idxs = repmat(struct('idx', zeros(1,l)), 1, nsecs);
0104     idxs = zeros(nsecs,l);
0105     nsecs = 0;
0106     start = 1;
0107     while start + l - 1 <= nx
0108         idxs(1+nsecs,:) = start:start+lm1;
0109         start = floor(start + l*Xolap);
0110         nsecs = nsecs + 1;
0111     end
0112     Sxy = zeros(nsecs,1);
0113     i2   = zeros(nsecs,1);
0114     xi   = zeros(l,1);
0115     xo   = zeros(l,1);
0116
0117     %----- Process the channels
0118
0119     % Loop over input channels
0120     for j=1:nc
0121         % loop over output channels
0122         for k=1:nc
0123             % compute lower triangle of matrix
0124             if j<k
0125
0126                 % do segments
0127                 Sxx    = 0;
0128                 Syy    = 0;
0129                 Sxy(:) = 0;
0130                 i2(:)  = 0;
0131
0132                 % Loop over segments
0133                 for s=1:nsecs
0134                     % get segments
0135                     xi(:) = X(idxs(s,:),j);
0136                     xo(:) = X(idxs(s,:),k);
0137
0138                     % detrend segments
0139                     eval(detrender);
0140
0141                     % compute DFT
0142                     ti    = C*xi;
0143                     to    = C*xo;
0144                     cto   = conj(to);
0145                     Sxy(s) = cto*ti;
0146                     Sxx    = Sxx + ti*conj(ti);

```

```

0147         Syy      = Syy + to*cto;
0148
0149     end % End segment loop
0150     disp(['-processed ' num2str(nsegs) ' segments']);
0151
0152     S1      = sum(win.ws);
0153     S12     = S1*S1;
0154     S2      = win.win*win.win';
0155     ENBW(j,k,fi) = fs*S2/S12;
0156     Txy(j,k,fi) = (abs(sum(Sxy))^2)/(Sxx*Syy);
0157
0158     %----- compute variance
0159     if computeVariance
0160         A      = 0;
0161         scale = fs*S2;
0162         for s=1:nsegs
0163             S    = (Sxy(s)/scale-Txy(j,k,fi))^2;
0164             A    = A + S * i2(s);
0165         end
0166         Sig(j,k,fi) = A/sum(i2);
0167     end % End compute variance
0168
0169     end % End if j<k
0170 end % End loop over output channels
0171 end % End loop over input channels
0172 end % End loop over frequencies
0173
0174 % Fill the holes
0175 aoOnes = ones(1,nf);
0176 for j=1:nc
0177     for k=1:nc
0178         if j == k % the answer is 1
0179             Txy(j,k,:) = aoOnes;
0180         end
0181         if j>k
0182             Txy(j,k,:) = Txy(k,j,:);
0183         if computeVariance
0184             Sig(j,k,:) = Sig(k,j,:);
0185         end
0186         end
0187     end
0188 end
0189
0190
0191 varargout{1} = Txy;
0192 if nargout == 2
0193     varargout{2} = ENBW;
0194 elseif nargout == 3
0195     varargout{3} = Sig;
0196     varargout{2} = ENBW;
0197 end

```


[Home](#) > [classes](#) > [@xydata](#) > save.m

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 1.1 2007/08/29 18:42:18 hewitson Exp \$

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

```
>> pl = save(xydata, 'Params')
```

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE an xydata object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for xydata objects.
0008 %
0009 % CALL:          Save an xydata object as a .mat file.
0010 %                >> save(d, 'blah.mat')
0011 %                Save an xydata object as an XML file.
0012 %                >> save(d, 'blah.xml')
0013 %                Save an xydata object to file specified in plist.
0014 %                >> save(d, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:       $Id: save.m,v 1.1 2007/08/29 18:42:18 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(xydata, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'xydata') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041 ALGONAME = mfilename;
0042 VERSION  = '$Id: save.m,v 1.1 2007/08/29 18:42:18 hewitson Exp $';

```

```
0043
0044 if nargin ~= 2
0045     help mfilename;
0046     error('### incorrect inputs');
0047 end
0048
0049 % get object
0050 obj = varargin{1};
0051 if isa(varargin{2}, 'plist')
0052     pl = varargin{2};
0053 elseif ischar(varargin{2})
0054     pl = plist(param('filename', varargin{2}));
0055 else
0056     help mfilename;
0057     error('### incorrect inputs');
0058 end
0059
0060 % Save object
0061 ltpda_saveobj(obj, pl);
0062
0063 % Get default params
0064 function plo = getDefaultPL()
0065
0066 disp('* creating default plist...');
0067 plo = plist();
0068 plo = append(plo, param('filename', 'object.xml'));
0069 disp('* done. ');
0070
0071
0072 % END
```


[Home](#) > [classes](#) > [@provenance](#) > xmladd.m

xmladd

PURPOSE

XMLADD adds specwin object to the given node of the xml input

SYNOPSIS

```
function xml = xmladd(p, xml, nodename, node)
```

DESCRIPTION

XMLADD adds specwin object to the given node of the xml input DOM.

M Hewitson 14-02-07

\$Id: xmladd.m,v 1.1 2007/08/29 18:39:05 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [xml](#) XML convert a provenance object to an DOM XML object.
- [xmladd](#) XMLADD adds specwin object to the given node of the xml input

This function is called by:

- [xml](#) XML convert a provenance object to an DOM XML object.
- [xmladd](#) XMLADD adds specwin object to the given node of the xml input

SOURCE CODE

```
0001 function xml = xmladd(p, xml, nodename, node)
0002
```

```
0003 % XMLADD adds specwin object to the given node of the xml input
0004 % DOM.
0005 %
0006 % M Hewitson 14-02-07
0007 %
0008 % $Id: xmladd.m,v 1.1 2007/08/29 18:39:05 hewitson Exp $
0009 %
0010
0011 el = xml.docNode.createElement(nodename);
0012
0013 % Now add object
0014 elc = xml.docNode.createElement('provenance');
0015
0016 % add creator
0017 elcc = xml.docNode.createElement('creator');
0018 elcc.appendChild(xml.docNode.createTextNode(p.creator));
0019 elc.appendChild(elcc);
0020
0021 % add created
0022 xml = xmladd(p.created, xml, 'created', elc);
0023
0024 % add ip
0025 elcc = xml.docNode.createElement('ip');
0026 elcc.appendChild(xml.docNode.createTextNode(p.ip));
0027 elc.appendChild(elcc);
0028
0029 % add hostname
0030 elcc = xml.docNode.createElement('hostname');
0031 elcc.appendChild(xml.docNode.createTextNode(p.hostname));
0032 elc.appendChild(elcc);
0033
0034 % add OS
0035 elcc = xml.docNode.createElement('os');
0036 elcc.appendChild(xml.docNode.createTextNode(p.os));
0037 elc.appendChild(elcc);
0038
0039 % add matlab_version
0040 elcc = xml.docNode.createElement('matlab_version');
0041 elcc.appendChild(xml.docNode.createTextNode(p.matlab_version));
0042 elc.appendChild(elcc);
0043
0044 % add sigproc_version
0045 elcc = xml.docNode.createElement('sigproc_version');
```

```
0046 elcc.appendChild(xml.docNode.createTextNode(p.sigproc_version));
0047 elc.appendChild(elcc);
0048
0049 % add ltpda_version
0050 elcc = xml.docNode.createElement('ltpda_version');
0051 elcc.appendChild(xml.docNode.createTextNode(p.ltpda_version));
0052 elc.appendChild(elcc);
0053
0054 el.appendChild(elc);
0055 node.appendChild(el);
0056
0057
0058
0059 % END
```

dzip

PURPOSE

DZIP - losslessly compress data into smaller memory space

SYNOPSIS

function Z = dzip(M)

DESCRIPTION

DZIP - losslessly compress data into smaller memory space

USAGE:

Z = dzip(M)

VARIABLES:

M = variable to compress

Z = compressed output

- NOTES:
- (1) The input variable M can be a scalar, vector, matrix, or n-dimensional matrix
 - (2) The input variable must be a non-complex and full (meaning matrices declared as type "sparse" are not allowed)
 - (3) Permitted input types include: double, single, logical, char, int8, uint8, int16, uint16, int32, uint32, int64, and uint64.
 - (4) In testing, DZIP compresses several megabytes of data per second.
 - (5) In testing, random matrices of type double compress to about 75% of their original size. Sparsely populated matrices or matrices with regular structure can compress to less than 1% of their original size. The realized compression ratio is heavily dependent on the data.
 - (6) Variables originally occupying very little memory (less than about half of one kilobyte) are handled correctly, but the compression requires some overhead and may actually increase the storage size of such small data sets. One exception to this rule is noted below.
 - (7) LOGICAL variables are compressed to a small fraction of their original sizes.
 - (8) The DUNZIP function decompresses the output of this function and restores the original data, including size and class type.

- (9) This function uses the public domain ZLIB Deflater algorithm.
- (10) Carefully tested, but no warranty; use at your own risk.
- (11) Michael Kleder, Nov 2005

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```

0001 function Z = dzip(M)
0002 % DZIP - losslessly compress data into smaller memory space
0003 %
0004 % USAGE:
0005 % Z = dzip(M)
0006 %
0007 % VARIABLES:
0008 % M = variable to compress
0009 % Z = compressed output
0010 %
0011 % NOTES: (1) The input variable M can be a scalar, vector, matrix, or
0012 %          n-dimensional matrix
0013 %          (2) The input variable must be a non-complex and full (meaning
0014 %              matrices declared as type "sparse" are not allowed)
0015 %          (3) Permitted input types include: double, single, logical,
0016 %              char, int8, uint8, int16, uint16, int32, uint32, int64,
0017 %              and uint64.
0018 %          (4) In testing, DZIP compresses several megabytes of data per
0019 %              second.
0020 %          (5) In testing, random matrices of type double compress to about
0021 %              75% of their original size. Sparsely populated matrices or
0022 %              matrices with regular structure can compress to less than
0023 %              1% of their original size. The realized compression ratio
0024 %              is heavily dependent on the data.
0025 %          (6) Variables originally occupying very little memory (less than
0026 %              about half of one kilobyte) are handled correctly, but
0027 %              the compression requires some overhead and may actually
0028 %              increase the storage size of such small data sets.
0029 %              One exception to this rule is noted below.
0030 %          (7) LOGICAL variables are compressed to a small fraction of
0031 %              their original sizes.
0032 %          (8) The DUNZIP function decompresses the output of this function
0033 %              and restores the original data, including size and class type.
0034 %          (9) This function uses the public domain ZLIB Deflater algorithm.
0035 %          (10) Carefully tested, but no warranty; use at your own risk.
0036 %          (11) Michael Kleder, Nov 2005

```

```
0037
0038 s = size(M);
0039 c = class(M);
0040 cn = strmatch(c,{'double','single','logical','char','int8','uint8',...
0041     'int16','uint16','int32','uint32','int64','uint64'});
0042 if cn == 3 | cn == 4
0043     M=uint8(M);
0044 end
0045 M=typecast(M(:),'uint8');
0046 M=[uint8(cn);uint8(length(s));typecast(s(:),'uint8');M(:)];
0047 f=java.io.ByteArrayOutputStream();
0048 g=java.util.zip.DeflaterOutputStream(f);
0049 g.write(M);
0050 g.close;
0051 Z=typecast(f.toByteArray,'uint8');
0052 f.close;
0053 return
```

[Home](#) > [m](#) > [helper](#) > ltpda_mat2str.m

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 1.4 2007/10/26 14:57:36 hewitson Exp \$

HISTORY: 26-07-2007 Diepholz
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function str = ltpda_mat2str(number)
0002 % LTPDA_MAT2STR overloads the mat2str operator to set the precision at a
0003 % central place.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
```

```

0006 % DESCRIPTION: LTPDA_MAT2STR overloads the mat2str operator to set the
0007 %                precision at a central place.
0008 %
0009 % CALL:          str = ltpda_mat2str(number);
0010 %                str = ltpda_mat2str(matrix);
0011 %
0012 % VERSION:       $Id: ltpda_mat2str.m,v 1.4 2007/10/26 14:57:36 hewitson Exp $
0013 %
0014 % HISTORY:       26-07-2007 Diepholz
0015 %                Creation
0016 %
0017 %%%%%%%%%%%%%%%
0018
0019 MAX_PRECISION = 20;
0020
0021 s = size(number);
0022 if s(1) ~= 1 && s(2) ~= 1
0023     str = mat2str(number, MAX_PRECISION);
0024 else
0025
0026     % For vectors it is faster to use sprintf directly
0027     if s(1) ~= s(2)
0028         str = '[';
0029     else
0030         str = '';
0031     end
0032     if s(1) > s(2)
0033         str = [str sprintf('%.17g;', number)];
0034     else
0035         str = [str sprintf('%.17g ', number)];
0036     end
0037     if s(1) ~= s(2)
0038         str = [str(1:end-1) ']];
0039     else
0040         str = str(1:end-1);
0041     end
0042 end
0043
0044
0045

```


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 1.1 2007/12/24 19:47:06 hewitson Exp \$

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

>> pl = save(xyzdata, 'Params')

HISTORY: 24-12-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SUBFUNCTIONS

- `function plo = getDefaultPL()`

SOURCE CODE

```

0001 function varargout = save(varargin)
0002
0003 % SAVE an xyzdata object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for xyzdata objects.
0008 %
0009 % CALL:          Save an xyzdata object as a .mat file.
0010 %                >> save(d, 'blah.mat')
0011 %                Save an xyzdata object as an XML file.
0012 %                >> save(d, 'blah.xml')
0013 %                Save an xyzdata object to file specified in plist.
0014 %                >> save(d, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:       $Id: save.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(xyzdata, 'Params')
0024 %
0025 % HISTORY: 24-12-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029 %
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041 ALGONAME = mfilename;
0042 VERSION  = '$Id: save.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $';
0043
0044 if nargin ~= 2
0045     help mfilename;

```

```
0046     error('### incorrect inputs');
0047 end
0048
0049 % get object
0050 obj = varargin{1};
0051 if isa(varargin{2}, 'plist')
0052     pl = varargin{2};
0053 elseif ischar(varargin{2})
0054     pl = plist(param('filename', varargin{2}));
0055 else
0056     help mfilename;
0057     error('### incorrect inputs');
0058 end
0059
0060 % Save object
0061 ltpda_saveobj(obj, pl);
0062
0063 % Get default params
0064 function plo = getDefaultPL()
0065
0066 disp('* creating default plist...');
0067 plo = plist();
0068 plo = append(plo, param('filename', 'object.xml'));
0069 disp('* done. ');
0070
0071
0072 % END
```

[Home](#) > [classes](#) > [@pzmodel](#) > xmladd.m

xmladd

PURPOSE

XMLADD add a pzmodel object to the given node of the input XML DOM

SYNOPSIS

```
function xml = xmladd(ps, xml, nodename, node)
```

DESCRIPTION

XMLADD add a pzmodel object to the given node of the input XML DOM object.

M Hewitson 14-02-07

\$Id: xmladd.m,v 1.3 2007/10/15 17:16:57 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [get](#) GET get a pole/zero model property.
- [xml](#) XML convert a pzmodel object to an DOM XML object.
- [xmladd](#) XMLADD add a pzmodel object to the given node of the input XML DOM

This function is called by:

- [xml](#) XML convert a pzmodel object to an DOM XML object.
- [xmladd](#) XMLADD add a pzmodel object to the given node of the input XML DOM

SOURCE CODE

```
0001 function xml = xmladd(ps, xml, nodename, node)
0002
0003 % XMLADD add a pzmodel object to the given node of the input XML DOM
0004 % object.
0005 %
0006 % M Hewitson 14-02-07
0007 %
0008 % $Id: xmladd.m,v 1.3 2007/10/15 17:16:57 ingo Exp $
```

```
0009 %
0010
0011 el = xml.docNode.createElement(nodename);
0012
0013 for j=1:length(ps)
0014     p = ps(j);
0015
0016     elc = xml.docNode.createElement('pzmodel');
0017
0018     % Add gain, poles, zeros
0019
0020     % add name
0021     elcc = xml.docNode.createElement('name');
0022     elcc.appendChild(xml.docNode.createTextNode(get(p, 'name')));
0023     elc.appendChild(elcc);
0024
0025     % add gain
0026     elcc = xml.docNode.createElement('gain');
0027     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(get(p, 'gain'))));
0028     elc.appendChild(elcc);
0029
0030     % add Created
0031     xml = xmladd(p.created, xml, 'created', elc);
0032
0033     % add version
0034     elcc = xml.docNode.createElement('version');
0035     elcc.appendChild(xml.docNode.createTextNode(p.version));
0036     elc.appendChild(elcc);
0037
0038     % add poles
0039     ps = get(p, 'poles');
0040     if ~isempty(ps)
0041         xml = xmladd(ps, xml, 'poles', elc);
0042     end
0043
0044     % add zeros
0045     ps = get(p, 'zeros');
0046     if ~isempty(ps)
0047         xml = xmladd(ps, xml, 'zeros', elc);
0048     end
0049     el.appendChild(elc);
0050 end
0051
0052 node.appendChild(el);
0053
0054
0055
0056 % END
```


[Home](#) > [m](#) > [analyses](#) > [stuttgart](#) > e2e2ao.m

e2e2ao

PURPOSE

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

SYNOPSIS

This is a script file.

DESCRIPTION

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
convert .mat data to aos  
  
veronika 05/07/2007  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [test_dkalman](#) %%%%%%%%%
%%%%%%%%%

SOURCE CODE

```
0001 %%%%%%%%%  
0002 % convert .mat data to aos  
0003 %  
0004 % veronika 05/07/2007  
0005 %%%%%%%%%
```

```
0006
0007 clear all;
0008
0009 load('input_meas_kalm.mat');
0010
0011 fs    = 1;
0012 x12   = q_proc_x120MS_1Hz(1:50000);
0013 x1    = q_proc_x10MS_1Hz(1:50000);
0014 fsus  = Fsus_highpass_50_detr(1:50000);
0015
0016 % Make X12
0017 nsecs = length(x12)/fs;
0018 t      = linspace(0, nsecs-1/fs, nsecs*fs);
0019 out    = [t ; x12.'].';
0020 save 'x12.txt' out -ASCII -DOUBLE -TABS
0021 X12 = ao('x12.txt');
0022 save(X12, 'X12.xml');
0023
0024 % Make X1
0025 nsecs = length(x1)/fs;
0026 t      = linspace(0, nsecs-1/fs, nsecs*fs);
0027 out    = [t ; x1.'].';
0028 save 'x1.txt' out -ASCII -DOUBLE -TABS
0029 X1     = ao('x1.txt');
0030 save(X1, 'X1.xml');
0031
0032 % Make Fsus
0033 nsecs = length(fsus)/fs;
0034 t      = linspace(0, nsecs-1/fs, nsecs*fs);
0035 out    = [t ; fsus.'].';
0036 save 'fsus.txt' out -ASCII -DOUBLE -TABS
0037 Fsus   = ao('fsus.txt');
0038 save(Fsus, 'Fsus.xml');
0039
0040
```


[Home](#) > [m](#) > [helper](#) > ltpda_matchvectors.m

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

```

0001 function [x, y, fs] = ltpda_matchvectors(x, y, xfs, yfs)
0002 % LTPDA_RESAMPLE make the two vectors have the same length.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_RESAMPLE make the two vectors have the same length and
0007 %               sample rate by down-sampling the one with the higher samplerate.
0008 %               Vectors are also truncated to the length of the shorter vector.
0009 %
0010 % CALL:         [xo, yo, fs] = mresample(x, y, xfs, yfs)
0011 %
0012 % INPUTS:       x    - data vector
0013 %               y    - data vector
0014 %               xfs   - sample rate of x
0015 %               yfs   - sample rate of y
0016 %
0017 % OUTPUTS:      x    - new data vector
0018 %               y    - new data vector
0019 %               fs    - new sample rate of x AND y
0020 %
0021 % VERSION: $Id: ltpda_matchvectors.m,v 1.3 2007/08/14 09:25:24 ingo Exp $
0022 %
0023 % HISTORY: 26-01-2007 M Hewitson
0024 %           Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0027 %
0028 % check the sample rates
0029 if yfs > xfs
0030     dv_disp('!!! vectors are not equal sample rates. Down-sampling y. !!!');
0031     y = resample(y, xfs, yfs);
0032     fs = xfs;
0033 end
0034
0035 if xfs > yfs
0036     dv_disp('!!! vectors are not equal sample rates. Down-sampling x. !!!');
0037     x = resample(x,yfs,xfs);
0038     fs = yfs;
0039 end
0040
0041 if xfs == yfs
0042     fs = xfs;
0043 end
0044
0045 % now check the vector lengths
0046 nx = length(x);
0047 ny = length(y);
0048
0049 if nx > ny
0050     dv_disp('!!! vectors are not equal lengths. Truncating x. !!!');
0051     x = x(1:ny);

```

```
0052 end
0053 if ny > nx
0054     dv_disp('!!! vectors are not equal lengths. Truncating y. !!!');
0055     y = y(1:nx);
0056 end
0057
```

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 1.1 2007/08/29 18:42:48 hewitson Exp \$

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

>> pl = save(zero, 'Params')

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

```

0001 function varargout = save(varargin)
0002
0003 % SAVE a zero object to file.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVE overloads the save method for zero objects.
0008 %
0009 % CALL:          Save a zero object as a .mat file.
0010 %                >> save(z, 'blah.mat')
0011 %                Save a zero object as an XML file.
0012 %                >> save(z, 'blah.xml')
0013 %                Save a zero object to file specified in plist.
0014 %                >> save(z, plist)
0015 %
0016 % Parameters: 'filename' - name of the file
0017 %
0018 % VERSION:      $Id: save.m,v 1.1 2007/08/29 18:42:48 hewitson Exp $
0019 %
0020 % The following call returns a parameter list object that contains the
0021 % default parameter values:
0022 %
0023 % >> pl = save(zero, 'Params')
0024 %
0025 % HISTORY: 28-08-07 M Hewitson
0026 %          Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(varargin{1}, 'zero') && ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             varargout{1} = getDefaultPL();
0036             return
0037         end
0038     end
0039 end
0040
0041 ALGONAME = mfilename;
0042 VERSION  = '$Id: save.m,v 1.1 2007/08/29 18:42:48 hewitson Exp $';

```

```
0043
0044 if nargin ~= 2
0045     help mfilename;
0046     error('### incorrect inputs');
0047 end
0048
0049 % get object
0050 obj = varargin{1};
0051 if isa(varargin{2}, 'plist')
0052     pl = varargin{2};
0053 elseif ischar(varargin{2})
0054     pl = plist(param('filename', varargin{2}));
0055 else
0056     help mfilename;
0057     error('### incorrect inputs');
0058 end
0059
0060 % Save object
0061 ltpda_saveobj(obj, pl);
0062
0063 % Get default params
0064 function plo = getDefaultPL()
0065
0066 disp('* creating default plist...');
0067 plo = plist();
0068 plo = append(plo, param('filename', 'object.xml'));
0069 disp('* done. ');
0070
0071
0072 % END
```

[Home](#) > [classes](#) > [@specwin](#) > xmladd.m

xmladd

PURPOSE

XMLADD adds specwin object to the given node of the xml input

SYNOPSIS

function xml = xmladd(p, xml, nodename, node)

DESCRIPTION

XMLADD adds specwin object to the given node of the xml input DOM.

M Hewitson 14-02-07

\$Id: xmladd.m,v 1.3 2008/01/11 14:05:13 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [xml](#) XML convert a specwin object to an DOM XML object.

SOURCE CODE

```
0001 function xml = xmladd(p, xml, nodename, node)
0002
0003 % XMLADD adds specwin object to the given node of the xml input
0004 % DOM.
0005 %
0006 % M Hewitson 14-02-07
0007 %
0008 % $Id: xmladd.m,v 1.3 2008/01/11 14:05:13 ingo Exp $
0009 %
0010
0011 el = xml.docNode.createElement(nodename);
0012
0013 % Now add object
```

```
0014 elc = xml.docNode.createElement('specwin');
0015
0016 % add created
0017 elcc = xml.docNode.createElement('created');
0018 elcc.appendChild(xml.docNode.createTextNode(p.created));
0019 elc.appendChild(elcc);
0020
0021 % add version
0022 elcc = xml.docNode.createElement('version');
0023 elcc.appendChild(xml.docNode.createTextNode(p.version));
0024 elc.appendChild(elcc);
0025
0026 % add name
0027 elcc = xml.docNode.createElement('name');
0028 elcc.appendChild(xml.docNode.createTextNode(p.name));
0029 elc.appendChild(elcc);
0030
0031 % add psll
0032 elcc = xml.docNode.createElement('psll');
0033 elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(p.psll)));
0034 elc.appendChild(elcc);
0035
0036 % add N
0037 elcc = xml.docNode.createElement('n');
0038 elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(length(p.win))));
0039 elc.appendChild(elcc);
0040
0041
0042 el.appendChild(elc);
0043 node.appendChild(el);
0044
0045
0046
0047 % END
```


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 adjustments

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```

0001 function hout = ltpda_msuptitle(str)
0002 % LTPDA_MSUPTITLE Puts a title above all subplots.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_MSUPTITLE Puts a title above all subplots.
0007 %               MSUPTITLE('text') adds text to the top of the figure
0008 %               above all subplots (a "super title"). Use this function
0009 %               after all subplot commands.
0010 %
0011 % CALL:         ltpda_xaxis(x1,x2)
0012 %
0013 % REMARK:       If the figure or axis units are non-default, this will break.
0014 %
0015 % VERSION:      $Id: ltpda_msuptitle.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
0016 %
0017 % HISTORY:      15-06-1995 Drea Thomas (drea@mathworks.com)
0018 %               Creation.
0019 %               13-12-2000 John Cristion
0020 %               Modified.
0021 %               13-03-2004 Mark Histed (histed@mit.edu)
0022 %               Fix disappearing legend on last plot
0023 %               xx-xx-xxxx M Hewitson
0024 %               Just a copy and rename with some small adjustments
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0027
0028 % Parameters used to position the supertitle.
0029
0030 % Amount of the figure window devoted to subplots
0031 plotregion = .92;
0032
0033 % Y position of title in normalized coordinates
0034 titleypos = .95;
0035
0036 % Fontsize for supertitle
0037 %fs = get(gcf,'defaultaxesfontsize')+4;
0038
0039 fs = get(gcf,'defaultaxesfontsize');
0040
0041 % Fudge factor to adjust y spacing between subplots
0042 fudge=1;
0043
0044 haold = gca;
0045 figunits = get(gcf,'units');
0046
0047 % Get the (approximate) difference between full height (plot + title
0048 % + xlabel) and bounding rectangle.
0049
0050 if (~strcmp(figunits,'pixels')),
0051     set(gcf,'units','pixels');

```

```

0052         pos = get(gcf,'position');
0053         set(gcf,'units',figunits);
0054     else
0055         pos = get(gcf,'position');
0056     end
0057     ff = (fs-4)*1.27*5/pos(4)*fudge;
0058
0059     % The 5 here reflects about 3 characters of height below
0060     % an axis and 2 above. 1.27 is pixels per point.
0061
0062 % Determine the bounding rectangle for all the plots
0063
0064 % h = findobj('Type','axes');
0065
0066 % findobj is a 4.2 thing.. if you don't have 4.2 comment out
0067 % the next line and uncomment the following block.
0068
0069 % h = findobj(gcf,'Type','axes'); % Change suggested by Stacy J. Hills
0070
0071 % If you don't have 4.2, use this code instead
0072 ch = get(gcf,'children');
0073 h=[];
0074 for i=1:length(ch),
0075     if strcmp(get(ch(i),'type'),'axes'),
0076         h=[h,ch(i)];
0077     end
0078 end
0079
0080
0081 max_y=0;
0082 min_y=1;
0083
0084 oldtitle =0;
0085 for i=1:length(h),
0086     if (~strcmp(get(h(i),'Tag'),'suptitle')),
0087         pos=get(h(i),'pos');
0088         if (pos(2) < min_y), min_y=pos(2)-ff/5*3;end;
0089         if (pos(4)+pos(2) > max_y), max_y=pos(4)+pos(2)+ff/5*2;end;
0090     else
0091         oldtitle = h(i);
0092     end
0093 end
0094
0095 if max_y > plotregion,
0096     scale = (plotregion-min_y)/(max_y-min_y);
0097     for i=1:length(h),
0098         get(h(i),'Tag')
0099         if ~strcmp(get(h(i),'Tag'),'legend')
0100             pos = get(h(i),'position');
0101             pos(2) = (pos(2)-min_y)*scale+min_y;
0102             pos(4) = pos(4)*scale-(1-scale)*ff/5*3;
0103             set(h(i),'position',pos);

```

```
0104     end
0105 end
0106 end
0107
0108 np = get(gcf, 'nextplot');
0109 set(gcf, 'nextplot', 'add');
0110 if (oldtitle),
0111     delete(oldtitle);
0112 end
0113 ha=axes('pos',[0 1 1 1], 'visible', 'off', 'Tag', 'suptitle');
0114 ht=text(.5, titleypos-1, str); set(ht, 'horizontalalignment', 'center', 'fontsize',
fs);
0115 set(gcf, 'nextplot', np);
0116 axes(haold);
0117
0118 % fix legend if one exists
0119 legH = legend;
0120 if ~isempty(legH)
0121     axes(legH);
0122 end
0123
0124 if nargout,
0125     hout=ht;
0126 end
0127
```

[Home](#) > [classes](#) > [@time](#) > xmladd.m

xmladd

PURPOSE

XMLADD add a time object to the given node of the input XML DOM

SYNOPSIS

```
function xml = xmladd(tts, xml, nodename, node)
```

DESCRIPTION

XMLADD add a time object to the given node of the input XML DOM object.

Diepholz 23-07-2007

\$Id: xmladd.m,v 1.3 2007/10/26 14:39:16 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a time object into a string.
- [get](#) GET get time properties.
- [xml](#) XML convert a time object to an DOM XML object.
- [xmladd](#) XMLADD add a time object to the given node of the input XML DOM

This function is called by:

- [xml](#) XML convert a time object to an DOM XML object.
- [xmladd](#) XMLADD add a time object to the given node of the input XML DOM

SOURCE CODE

```
0001 function xml = xmladd(tts, xml, nodename, node)
0002
```

```

0003 % XMLADD add a time object to the given node of the input XML DOM
0004 % object.
0005 %
0006 % Diepholz 23-07-2007
0007 %
0008 % $Id: xmladd.m,v 1.3 2007/10/26 14:39:16 hewitson Exp $
0009 %
0010
0011 el = xml.docNode.createElement(nodename);
0012 for j=1:length(tts)
0013     tt = tts(j);
0014     elc = xml.docNode.createElement('time');
0015
0016     % add name
0017     elcc = xml.docNode.createElement('name');
0018     elcc.appendChild(xml.docNode.createTextNode(get(tt, 'name')));
0019     elc.appendChild(elcc);
0020
0021     % add utc_epoch_milli
0022     elcc = xml.docNode.createElement('utc_epoch_milli');
0023     elcc.appendChild(xml.docNode.createTextNode(num2str(get(tt,
'utc_epoch_milli'))));
0024     elc.appendChild(elcc);
0025
0026     % add timezone
0027     elcc = xml.docNode.createElement('timezone');
0028     timezone = get(tt, 'timezone');
0029     elcc.appendChild(xml.docNode.createTextNode(char(timezone.getID)));
0030     elc.appendChild(elcc);
0031
0032     % add timeformat
0033     xml = xmladd(get(tt, 'timeformat'), xml, 'timeformat', elc);
0034
0035     % add time_str
0036     elcc = xml.docNode.createElement('time_str');
0037     elcc.appendChild(xml.docNode.createTextNode(get(tt, 'time_str')));
0038     elc.appendChild(elcc);
0039
0040     % add created
0041     elcc = xml.docNode.createElement('created');
0042     elcc.appendChild(xml.docNode.createTextNode(get(tt, 'created')));
0043     elc.appendChild(elcc);
0044
0045     % add version
0046     elcc = xml.docNode.createElement('version');

```

```
0047     elcc.appendChild(xml.docNode.createTextNode(get(tt, 'version')));
0048     elc.appendChild(elcc);
0049
0050     el.appendChild(elc);
0051 end
0052 node.appendChild(el);
0053
0054
0055
0056 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [sigproc](#) > [frequency_domain](#) > ltpda_nfest.m

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

```

0001 function varargout = ltpda_nfest(varargin)
0002 % LTPDA_NFEST Estimate the noise floor of the given frequency-series A0.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_NFEST Estimate the noise floor of the given
0007 %               frequency-series A0. This essentially implements a running
0008 %               median filter with some outlier selection.
0009 %
0010 % CALL:        b = ltpda_nfest(a, pl)
0011 %
0012 % PARAMETERS:
0013 %   bw  - the bandwidth of the noise floor estimate [default: 20 samples]
0014 %   hc  - a cutoff to throw away outliers (0-1) [default: 0.8]
0015 %
0016 % The following call returns a parameter list object that contains the
0017 % default parameter values:
0018 %
0019 % >> pl = ltpda_nfest('Params')
0020 %
0021 %
0022 % VERSION: $Id: ltpda_nfest.m,v 1.7 2007/11/08 10:24:54 hewitson Exp $
0023 %
0024 % HISTORY: 14-05-07 M Hewitson
0025 %           Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028 %
0029 % Check if this is a call for parameters
0030 if nargin == 1
0031     in = char(varargin{1});
0032     if strcmp(in, 'Params')
0033         varargout{1} = getDefaultPL();
0034         return
0035     end
0036 end
0037
0038 % capture input variable names
0039 invars = {};
0040 for j=1:nargin
0041     if isa(varargin{j}, 'ao')

```

```

0042     invars = [invars cellstr(inputname(j))];
0043 end
0044 end
0045
0046 ALGONAME = mfilename;
0047 VERSION  = '$Id: ltpda_nfest.m,v 1.7 2007/11/08 10:24:54 hewitson Exp $';
0048
0049 % Get inputs
0050 as      = [];
0051 ps      = [];
0052 queries = [];
0053 for j=1:nargin
0054     if isa(varargin{j}, 'ao')
0055         as = [as varargin{j}];
0056     end
0057     if isa(varargin{j}, 'plist')
0058         ps = [ps varargin{j}];
0059     end
0060 end
0061
0062 Na = length(as);
0063 if isempty(as)
0064     error('### Please input at least one A0.');
```

```

0065 end
0066
0067 % Combine plists
0068 if ~isempty(ps)
0069     pl = combine(ps, getDefaultPL());
0070 else
0071     pl = getDefaultPL();
0072 end
0073
0074 % Get parameters from plist
0075 bw      = find(pl, 'bw');
0076 hc      = find(pl, 'hc');
0077
0078 % Loop over input A0s
0079 bo = [];
0080 for j=1:Na
0081
0082     a = as(j);
0083     d = a.data;
0084
0085     if isa(d, 'fsdata')
0086
0087         nf = nfest(d.y, bw, hc);
0088
0089     else
0090         error('### I can only find lines in frequency-series A0s.');
```

```

0091     end
0092
0093     %----- Make output A0

```

```

0094
0095 % make fsdata
0096 d = set(d, 'y', nf);
0097 d = set(d, 'name', sprintf('nfest(%s)', d.name));
0098
0099 % create new output history
0100 h = history(ALGONAME, VERSION, pl, a.hist);
0101 h = set(h, 'invars', invars);
0102
0103 % make output analysis object
0104 b = ao(d, h);
0105
0106 % set name
0107 % name for this object
0108 if isempty(invars{j})
0109     n1 = a.name;
0110 else
0111     n1 = invars{j};
0112 end
0113 b = setnh(b, 'name', sprintf('nfest(%s)', n1));
0114
0115 % Add to output array
0116 bo = [bo b];
0117
0118 end
0119
0120 varargout{1} = bo;
0121
0122 %-----
0123 % get default parameter list
0124 function pl = getDefaultPL()
0125
0126 pl = plist();
0127 pl = append(pl, param('bw', 20));
0128 pl = append(pl, param('hc', 0.8));
0129
0130
0131 %-----
0132 % compute noise floor estimate
0133 function nf = nfest(pxx, bw, hc)
0134
0135 N = length(pxx);
0136 nf = [];
0137
0138 for j=1:N
0139
0140     % Determine the interval we are looking in
0141     interval = j-bw/2:j+bw/2;
0142     idx = find(interval<=0);
0143     interval(idx)=1;
0144     idx = find(interval>N);
0145     interval(idx)=N;

```

```
0146
0147 % calculate median value of interval
0148 % after throwing away outliers
0149 trial = sort(pxx(interval));
0150 b = round(hc*length(trial));
0151 nf(j) = median(trial(1:b));
0152
0153 end
0154
0155 nf = nf(:);
0156
0157 % END
0158
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@timeformat](#) > xmladd.m

xmladd

PURPOSE

XMLADD add a time object to the given node of the input XML DOM

SYNOPSIS

function xml = xmladd(tf, xml, nodename, node)

DESCRIPTION

XMLADD add a time object to the given node of the input XML DOM object.

Diepholz 23-07-2007

\$Id: xmladd.m,v 1.1 2007/08/29 18:40:52 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [xml](#) XML convert a timeformat object to an DOM XML object.

SOURCE CODE

```
0001 function xml = xmladd(tf, xml, nodename, node)
0002
0003 % XMLADD add a time object to the given node of the input XML DOM
0004 % object.
0005 %
0006 % Diepholz 23-07-2007
0007 %
0008 % $Id: xmladd.m,v 1.1 2007/08/29 18:40:52 hewitson Exp $
0009 %
0010
0011 el = xml.docNode.createElement(nodename);
0012 elc = xml.docNode.createElement('timeformat');
0013
```

```
0014
0015 % add format_str
0016 elcc = xml.docNode.createElement('format_str');
0017 elcc.appendChild(xml.docNode.createTextNode(tf.format_str));
0018 elc.appendChild(elcc);
0019
0020 % add format_nr
0021 elcc = xml.docNode.createElement('format_nr');
0022 elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(tf.format_nr)));
0023 elc.appendChild(elcc);
0024
0025 % add default_matlab_str
0026 elcc = xml.docNode.createElement('default_matlab_str');
0027 elcc.appendChild(xml.docNode.createTextNode(tf.default_matlab_str));
0028 elc.appendChild(elcc);
0029
0030 % add default_matlab_nr
0031 elcc = xml.docNode.createElement('default_matlab_nr');
0032 elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(tf.
default_matlab_nr)));
0033 elc.appendChild(elcc);
0034
0035 % add default_java_str
0036 elcc = xml.docNode.createElement('default_java_str');
0037 elcc.appendChild(xml.docNode.createTextNode(tf.default_java_str));
0038 elc.appendChild(elcc);
0039
0040 el.appendChild(elc);
0041 node.appendChild(el);
0042
0043
0044
0045 % END
```

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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(c1,c2, 'ao/name')

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 1.13 2007/11/27 13:18:08 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:

- **eq** EQ overloads the == operator for cdata objects.
- **ne** NE overloads the ~= operator for cdata objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002 % EQ overloads the == operator for cdata objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EQ overloads the == operator for cdata objects.
0007 %
0008 %           All fields are checked.
0009 %
0010 % CALL:      result = eq(c1,c2)
0011 %           result = eq(c1,c2, exc_list)
0012 %           result = eq(c1,c2, 'property1', 'property2')
0013 %           result = eq(c1,c2, 'class/property', 'class/property')
0014 %
0015 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0016 %           result = eq(c1,c2, 'ao/name')
0017 %
0018 % INPUTS:    c1,c2      - input cdata objects
0019 %           exc_list - exception list
0020 %           List of properties which are not checked.
0021 %
0022 % OUTPUTS:   If the two cdata objects are considered equal, result == 1,
0023 %           otherwise, result == 0.
0024 %
0025 % VERSION:   $Id: eq.m,v 1.13 2007/11/27 13:18:08 ingo Exp $
0026 %
0027 % HISTORY:   29-08-2007 M Hewitson
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION = '$Id: eq.m,v 1.13 2007/11/27 13:18:08 ingo Exp $';
0033
0034 % Check if this is a call for parameters
0035 if nargin == 2
0036     if isa(c1, 'cdata') && ischar(c2)
0037         in = char(c2);
0038         if strcmp(in, 'Params')
0039             result = plist;
0040             return

```



```
0041     elseif strcmp(in, 'Version')
0042         result = VERSION;
0043         return
0044     end
0045 end
0046 end
0047
0048 result = 1;
0049
0050 %% Check class
0051 if ~strcmp(class(c1), class(c2))
0052     result = 0;
0053     return
0054 end
0055
0056 fields = fieldnames(c1);
0057
0058 for ii = 1:length(fields)
0059     field = fields{ii};
0060
0061     ck_field = {field, ['cdata/' field]};
0062
0063     %% Check tag
0064     if ~(any(ismember(ck_field, varargin)))
0065
0066         if isobject(c1.(field))
0067             if ~eq(c1.(field), c2.(field), varargin{:})
0068                 result = 0;
0069                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0070                 return
0071             end
0072         else
0073             if ~isequalwithequalnans(c1.(field), c2.(field))
0074                 result = 0;
0075                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0076                 return
0077             end
0078         end
0079     end
0080 end
0081 end
0082
```

[Home](#) > [m](#) > [noisegenerator](#) > ltpda_noisegen.m

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](#) % get zeros and poles from pzmodels
- [nginit](#)
- [ngprop](#)
- [ngsetup](#) ALGONAME = mfilename;

This function is called by:

SUBFUNCTIONS

- [function plo = getDefaultPL\(\)](#)

SOURCE CODE

```

0001 function varargout = ltpda_noisegen(varargin)
0002
0003 % LTPDA_NOISEGEN uses pzmodel and franklin to generate a time-series
0004 % from a given pzmodel
0005 %
0006 % calls:
0007 % >> [b, pl1, pl2] = ltpda_noisegen(pl)
0008 % >> [b, pl1] = ltpda_noisegen(pl)
0009 % >> b = ltpda_noisegen(pl1, pl2)
0010 %
0011 % Inputs:
0012 %     for the first function call the parameter list has to contain:
0013 %     - nsecs
0014 %     - fs
0015 %     - pzmodel with gain
0016 %
0017 % b     - analysis object containing the resulting timeseries
0018 % pl1   - parameter list containing the last state vector y.
0019 % pl2   - parameter list containing the following parameters:
0020 %
0021 %         - Tinit - matrix to calculate initial state vector
0022 %         - Tprop - matrix to calculate propagation vector
0023 %         - E     - matrix to calculate propagation vector
0024 %         - num   - numerator coefficients
0025 %         - den   - denominator coefficients
0026 %         - gain
0027 %         - fs    - sampling frequency
0028 %         - nsecs - number of seconds
0029 %
0030 % The following call returns a parameter list object that contains the
0031 % default parameter values:
0032 % >> pl = ltpda_noisegen('Params')
0033 %

```

```

0034 % A Monsky 24-07-07
0035 %
0036 % $Id: ltpda_noisegen.m,v 1.10 2007/11/09 14:07:39 ingo Exp $
0037 %
0038
0039 ALGONAME = mfilename;
0040 VERSION  = '$Id: ltpda_noisegen.m,v 1.10 2007/11/09 14:07:39 ingo Exp $';
0041
0042 % Check if this is a call for parameters
0043 if nargin == 1
0044     in = char(varargin{1});
0045     if strcmp(in, 'Params')
0046         varargout{1} = getDefaultPL();
0047         return
0048     end
0049 end
0050
0051 % capture input variable names
0052 invars = {};
0053 ps     = [];
0054 for j=1:nargin
0055     invars = [invars cellstr(inputname(j))];
0056     if isa(varargin{j}, 'plist')
0057         ps = [ps varargin{j}];
0058     end
0059 end
0060
0061 pl = combine(ps);
0062
0063 % look for all parameters
0064 pzmodel = find(pl, 'pzmodel');
0065 gain     = get(pzmodel, 'gain');
0066 fs       = find(pl, 'fs');
0067 nsecs    = find(pl, 'nsecs');
0068 state    = find(pl, 'state');
0069 Tinit    = find(pl, 'Tinit');
0070 Tprop    = find(pl, 'Tprop');
0071 E        = find(pl, 'E');
0072 num      = find(pl, 'num');
0073 den      = find(pl, 'den');
0074
0075 % get data out of aos
0076 if ~isempty(state)
0077     d      = state.data;
0078     state = d.y;
0079 end
0080 if ~isempty(Tinit)
0081     d      = Tinit.data;
0082     Tinit = d.y;
0083 end
0084 if ~isempty(Tprop)
0085     d      = Tprop.data;
0086     Tprop = d.y;

```

```

0087 end
0088 if ~isempty(E)
0089     d      = E.data;
0090     E = d.y;
0091 end
0092 if ~isempty(num)
0093     d      = num.data;
0094     num = d.y;
0095 end
0096 if ~isempty(den)
0097     d      = den.data;
0098     den = d.y;
0099 end
0100 lengden = length(den);
0101 % check calling state
0102 if isempty(pzmodel)
0103     if isempty(Tinit)
0104         error('### There is no Tinit matrix defined plist.');
```

end

```

0106     gain = find(pl, 'gain');
0107     % set state vector
0108     if isempty(state)
0109         disp('There is no pzmodel and no initial state vector given, so the initial
state vector will be calculated by using the given matrices.\\');
0110         % make initial state vector
0111         y = nginit(Tinit);
0112     else
0113         disp('State vector from input parameter list is used as input state.');
```

% use input state

```

0115     y = state;
0116 end
0117
0118 % propagate to make noise vector
0119 [x, yo] = ngprop(Tprop, E, num, y, fs*nsecs);
0120 state = yo;
0121 else
0122     % conversion
0123     disp('Filter coefficients are calculated from input pzmodel.');
```

[num, den] = ngconv(pzmodel);

```

0125     % create matrices
0126     disp('Matrices are calculated from evaluated denominator coefficients.');
```

[Tinit, Tprop, E] = ngsetup(den, fs);

```

0128
0129     % set state vector
0130     if isempty(state)
0131         disp('Since there is no given state vector it will be calculated.');
```

% make initial state vector

```

0133     y = nginit(Tinit);
0134     else
0135         disp('Since there is a state vector given it is used for further
calculations.');
```

% use input state

```

0136
```

```

0137     y = state;
0138 end
0139 % propagate to make noise vector
0140 [x, yo] = ngprop(Tprop, E, num, y, fs*nsecs);
0141 state = yo;
0142 end
0143
0144 %----- Build outputs
0145 % make a new history object
0146 h = history(ALGONAME, VERSION, pl);
0147 h = set(h, 'invars', invars);
0148 % build variables into data object
0149 t = x*gain;
0150 b = ao(tdata(t,fs),h);
0151 b = setnh(b, 'name', sprintf('noisegen(%s)', b.name));
0152
0153 state = ao(cdata(state),h);
0154 Tinit = ao(cdata(Tinit),h);
0155 Tprop = ao(cdata(Tprop),h);
0156 E = ao(cdata(E),h);
0157 num = ao(cdata(num),h);
0158 den = ao(cdata(den),h);
0159 if nargout == 2
0160     % build pl with 'state'
0161     pl1 = plist('state', state);
0162 end
0163
0164 if nargout == 3
0165     % build another pl with 'Tinit', 'Tprop', 'E', 'num', 'den'
0166     pl1 = plist('state', state);
0167     pl2 = plist('Tinit', Tinit, 'Tprop', Tprop, 'E', E, 'num', num, 'den', den, 'gain',
gain, 'fs', fs, 'nsecs', nsecs);
0168 end
0169
0170 %----- Pass outputs
0171
0172 if nargout == 1
0173     varargout{1} = b;
0174 end
0175
0176 if nargout == 2
0177     varargout{1} = b;
0178     varargout{2} = pl1;
0179 end
0180
0181 if nargout == 3
0182     varargout{1} = b;
0183     varargout{2} = pl1;
0184     varargout{3} = pl2;
0185 end
0186
0187
0188 % END

```

```
0189
0190 %-----
0191 % Get default params
0192 function plo = getDefaultPL()
0193
0194 disp('* creating default plist...');
0195 plo = plist();
0196 plo = append(plo,param('nsecs',1000));
0197 disp('* done. ');
0198
0199
0200 % END
0201
0202
0203
```

[Home](#) > [classes](#) > [@cdata](#) > set.m

set

PURPOSE

SET set a cdata property.

SYNOPSIS

function c = set(c, varargin)

DESCRIPTION

SET set a cdata property.

%%%

DESCRIPTION: SET set a cdata property.

```
CALL: c = set(c, 'name',      'my_name');
      c = set(c, 'y',        y);
      c = set(c, 'y',        [1 2 3]);
      c = set(c, 'vals',     [1 2 3]);
      c = set(c, 'x',        x);
      c = set(c, 'x',        {'a', 'c', 'c'});
      c = set(c, 'tags',     {'a', 'c', 'c'});
      c = set(c, 'xunits',   'sec');
      c = set(c, 'yunits',   'V');
      c = set(c, 'created',  time());
      c = set(c, 'version',  'cdata/set.m,v 1.7...');
```

VERSION: \$Id: set.m,v 1.9 2007/11/13 16:03:20 ingo Exp \$

HISTORY: 30-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a cdata object into a string.

This function is called by:

- `cdata` CData constant data object class constructor.

SOURCE CODE

```

0001 function c = set(c, varargin)
0002 % SET set a cdata property.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET set a cdata property.
0007 %
0008 % CALL: c = set(c, 'name',      'my_name');
0009 %       c = set(c, 'y',        y);
0010 %       c = set(c, 'y',        [1 2 3]);
0011 %       c = set(c, 'vals',     [1 2 3]);
0012 %       c = set(c, 'x',        x);
0013 %       c = set(c, 'x',        {'a', 'c', 'c'});
0014 %       c = set(c, 'tags',     {'a', 'c', 'c'});
0015 %       c = set(c, 'xunits',   'sec');
0016 %       c = set(c, 'yunits',   'V');
0017 %       c = set(c, 'created',   time());
0018 %       c = set(c, 'version',   'cdata/set.m,v 1.7...');
0019 %
0020 % VERSION: $Id: set.m,v 1.9 2007/11/13 16:03:20 ingo Exp $
0021 %
0022 % HISTORY: 30-01-07 M Hewitson
0023 %           Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026
0027 VERSION = '$Id: set.m,v 1.9 2007/11/13 16:03:20 ingo Exp $';
0028
0029 % Check if this is a call for parameters
0030 if nargin == 2
0031     if isa(c, 'cdata') && ischar(varargin{1})
0032         in = char(varargin{1});
0033         if strcmp(in, 'Params')
0034             c = plist;
0035             return
0036         elseif strcmp(in, 'Version')
0037             c = VERSION;
0038             return
0039         end
0040     end
0041 end
0042
0043
0044 if (nargout ~= 0)
0045
0046     propArgin = varargin;
0047

```

```

0048 while length(propArgin) >= 2
0049     prop = propArgin{1};
0050     val = propArgin{2};
0051     propArgin = propArgin(3:end);
0052     switch prop
0053         case 'name'
0054             c.name = val;
0055         case 'created'
0056             c.created = val;
0057         case 'version'
0058             c.version = val;
0059         case 'vals'
0060             c.y = val;
0061         case 'y'
0062             c.y = val;
0063         case 'tags'
0064             c.x = val;
0065         case 'x'
0066             c.x = val;
0067         case 'xunits'
0068             c.xunits = val;
0069         case 'yunits'
0070             c.yunits = val;
0071         otherwise
0072             error([val, ' is not a property of cdata class']);
0073     end
0074 end
0075
0076 else
0077     if ischar(varargin{2})
0078         error('### please use: %s = set(%s, '%s'', '%s'');', ...
0079             inputname(1), ...
0080             inputname(1), ...
0081             varargin{1}, ...
0082             varargin{2});
0083     elseif isnumeric(varargin{2})
0084         error('### please use: %s = set(%s, '%s'', %d);', ...
0085             inputname(1), ...
0086             inputname(1), ...
0087             varargin{1}, ...
0088             varargin{2});
0089     else
0090         error('### please use: %s = set(%s, 'key'', 'value'');', ...
0091             inputname(1), ...
0092             inputname(1));
0093     end
0094 end

```

[Home](#) > [classes](#) > [@timespan](#) > xmladd.m

xmladd

PURPOSE

XMLADD add a timespan object to the given node of the input XML DOM

SYNOPSIS

```
function xml = xmladd(tss, xml, nodename, node)
```

DESCRIPTION

XMLADD add a timespan object to the given node of the input XML DOM object.

Diepholz 23-07-2007

\$Id: xmladd.m,v 1.2 2007/10/16 17:37:21 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a timespan object into a string.
- [xml](#) XML convert a time object to an DOM XML object.
- [xmladd](#) XMLADD add a timespan object to the given node of the input XML DOM

This function is called by:

- [xml](#) XML convert a time object to an DOM XML object.
- [xmladd](#) XMLADD add a timespan object to the given node of the input XML DOM

SOURCE CODE

```
0001 function xml = xmladd(tss, xml, nodename, node)
0002
0003 % XMLADD add a timespan object to the given node of the input XML DOM
```

```

0004 % object.
0005 %
0006 % Diepholz 23-07-2007
0007 %
0008 % $Id: xmladd.m,v 1.2 2007/10/16 17:37:21 ingo Exp $
0009 %
0010
0011 el = xml.docNode.createElement(nodename);
0012 for j=1:length(tss)
0013     ts = tss(j);
0014     elc = xml.docNode.createElement('timespan');
0015
0016     % add name
0017     elcc = xml.docNode.createElement('name');
0018     elcc.appendChild(xml.docNode.createTextNode(ts.name));
0019     elc.appendChild(elcc);
0020
0021     % add start time
0022     xml = xmladd(ts.start, xml, 'start', elc);
0023
0024     % add stop time
0025     xml = xmladd(ts.end, xml, 'end', elc);
0026
0027     % add timeformat
0028     xml = xmladd(ts.timeformat, xml, 'timeformat', elc);
0029
0030     % add timezone
0031     elcc = xml.docNode.createElement('timezone');
0032     timezone = ts.timezone;
0033     elcc.appendChild(xml.docNode.createTextNode(char(timezone.getID)));
0034     elc.appendChild(elcc);
0035
0036     % add created
0037     xml = xmladd(ts.created, xml, 'created', elc);
0038
0039     % add version
0040     elcc = xml.docNode.createElement('version');
0041     elcc.appendChild(xml.docNode.createTextNode(ts.version));
0042     elc.appendChild(elcc);
0043
0044     el.appendChild(elc);
0045 end
0046 node.appendChild(el);
0047
0048

```

0049

0050 % END

Generated on Tue 22-Jan-2008 10:39:13 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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(c1,c2, 'ao/name')

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 1.7 2007/11/27 12:15:40 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:

- **eq** EQ overloads the == operator for fsdata objects.
- **ne** NE overloads the ~= operator for fsdata objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002
0003 % EQ overloads the == operator for fsdata objects.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: EQ overloads the == operator for fsdata objects.
0008 %
0009 %           All fields are checked.
0010 %
0011 % CALL:      result = eq(c1,c2)
0012 %           result = eq(c1,c2, exc_list)
0013 %           result = eq(c1,c2, 'property1', 'property2')
0014 %           result = eq(c1,c2, 'class/property', 'class/property')
0015 %
0016 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0017 %           result = eq(c1,c2, 'ao/name')
0018 %
0019 % INPUTS:    c1,c2      - input fsdata objects
0020 %           exc_list - exception list
0021 %           List of properties which are not checked.
0022 %
0023 % OUTPUTS:   If the two fsdata objects are considered equal, result == 1,
0024 %           otherwise, result == 0.
0025 %
0026 % VERSION:   $Id: eq.m,v 1.7 2007/11/27 12:15:40 ingo Exp $
0027 %
0028 % HISTORY:   29-08-2007 M Hewitson
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032
0033 VERSION = '$Id: eq.m,v 1.7 2007/11/27 12:15:40 ingo Exp $';
0034
0035 % Check if this is a call for parameters
0036 if nargin == 2
0037     if isa(c1, 'fsdata') && ischar(c2)
0038         in = char(c2);
0039         if strcmp(in, 'Params')
0040             result = plist;
0041             return

```

```

0042     elseif strcmp(in, 'Version')
0043         result = VERSION;
0044         return
0045     end
0046 end
0047 end
0048
0049 result = 1;
0050
0051 %% Check class
0052 if ~strcmp(class(c1), class(c2))
0053     result = 0;
0054     return
0055 end
0056
0057 fields = fieldnames(c1);
0058
0059 for ii = 1:length(fields)
0060     field = fields{ii};
0061
0062     ck_field = {field, ['fsdata/' field]};
0063
0064     %% Check tag
0065     if ~(any(ismember(ck_field, varargin)))
0066
0067         if isobject(c1.(field))
0068             if ~eq(c1.(field), c2.(field), varargin{:})
0069                 result = 0;
0070                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0071                 return
0072             end
0073         else
0074             if ~isequalwithhequalnans(c1.(field), c2.(field))
0075                 result = 0;
0076                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0077                 return
0078             end
0079         end
0080
0081     end
0082 end
0083
0084
0085
0086 % function result = eq(c1,c2, varargin)
0087 %
0088 % % EQ overloads the == operator for fsdata objects.
0089 % %
0090 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0091 % %
0092 % % DESCRIPTION: EQ overloads the == operator for fsdata objects.
0093 % %

```



```

0094 % %           The 'name', 'fs', 'enbw', 'xunits', 'yunits', and data values
are checked.
0095 % %
0096 % % CALL:      result = eq(c1,c2)
0097 % %
0098 % % INPUTS:     c1,c2 - input fsdata objects
0099 % %
0100 % % OUTPUTS:    If the two fsdata objects are considered equal, result == 1,
0101 % %              otherwise, result == 0.
0102 % %
0103 % % VERSION:    $Id: eq.m,v 1.7 2007/11/27 12:15:40 ingo Exp $
0104 % %
0105 % % HISTORY:     29-08-2007 M Hewitson
0106 % %              Creation
0107 % %
0108 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
0109 %
0110 % % Assume equality to begin with
0111 % result = 1;
0112 %
0113 % %% Check name
0114 % if ~ismember('name', varargin)
0115 %     if ~strcmp(c1.name, c2.name)
0116 %         result = 0;
0117 %         return
0118 %     end
0119 % end
0120 %
0121 % %% check fs
0122 % if ~ismember('fs', varargin)
0123 %     if c1.fs ~= c2.fs
0124 %         result = 0;
0125 %         return
0126 %     end
0127 % end
0128 %
0129 % %% check enbw
0130 % if ~ismember('enbw', varargin)
0131 %     if c1.enbw ~= c2.enbw
0132 %         result = 0;
0133 %         return
0134 %     end
0135 % end
0136 %
0137 % %% check xunits
0138 % if ~ismember('xunits', varargin)
0139 %     if ~strcmp(c1.xunits, c2.xunits)
0140 %         result = 0;
0141 %         return
0142 %     end
0143 % end
0144 %

```

```
0145 % %% check yunits
0146 % if ~ismember('yunits', varargin)
0147 %     if ~strcmp(c1.yunits, c2.yunits)
0148 %         result = 0;
0149 %         return
0150 %     end
0151 % end
0152 %
0153 % %% Check f data
0154 % if ~ismember('f', varargin)
0155 %     if length(c1.f) ~= length(c2.f)
0156 %         result = 0;
0157 %         return
0158 %     end
0159 %
0160 %     if c1.f ~= c2.f
0161 %         result = 0;
0162 %         return
0163 %     end
0164 % end
0165 %
0166 % %% Check xx data
0167 % if ~ismember('xx', varargin)
0168 %     if length(c1.xx) ~= length(c2.xx)
0169 %         result = 0;
0170 %         return
0171 %     end
0172 %
0173 %     if c1.xx ~= c2.xx
0174 %         result = 0;
0175 %         return
0176 %     end
0177 % end
```

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:

SOURCE CODE

```
0001 function str = ltpda_num2str(number)
0002 % LTPDA_NUM2STR uses sprintf to convert a data vector to a string with a fixed
0003 % precision.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: uses sprintf to convert a data vector to a string with a fixed
```

```
precision.  
0007 %  
0008 % CALL:          str = ltpda_num2str(number);  
0009 %  
0010 % VERSION:       $Id: ltpda_num2str.m,v 1.2 2007/10/26 14:39:54 hewitson Exp $  
0011 %  
0012 % HISTORY:       18-10-2007 Hewitson  
0013 %                 Creation  
0014 %  
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
0016  
0017  
0018 str = sprintf('%.17g ', number);  
0019
```

[Home](#) > [classes](#) > [@fsdata](#) > set.m

set

PURPOSE

SET set a fsdata property.

SYNOPSIS

function fsd = set(fsd, varargin)

DESCRIPTION

SET set a fsdata property.

%%%

DESCRIPTION: SET set a fsdata property.

```
CALL: fsd = set(fsd, 'name',      'my_name');
      fsd = set(fsd, 'fs',       fs);
      fsd = set(fsd, 'x',        x);
      fsd = set(fsd, 'enbw',     enbw);
      fsd = set(fsd, 'navs',     navs);
      fsd = set(fsd, 'y',        y);
      fsd = set(fsd, 'xunits',   'Hz');
      fsd = set(fsd, 'yunits',   'Volts');
      fsd = set(fsd, 'created',   time());
      fsd = set(fsd, 'version',  'fsdata/set.m,v 1.7...');
```

VERSION: \$Id: set.m,v 1.13 2007/11/13 16:03:20 ingo Exp \$

HISTORY: 30-01-07 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

```

0001 function fsd = set(fsd, varargin)
0002 % SET set a fsdata property.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET set a fsdata property.
0007 %
0008 % CALL: fsd = set(fsd, 'name',      'my_name');
0009 %         fsd = set(fsd, 'fs',      fs);
0010 %         fsd = set(fsd, 'x',      x);
0011 %         fsd = set(fsd, 'enbw',    enbw);
0012 %         fsd = set(fsd, 'navs',    navs);
0013 %         fsd = set(fsd, 'y',      y);
0014 %         fsd = set(fsd, 'xunits',  'Hz');
0015 %         fsd = set(fsd, 'yunits',  'Volts');
0016 %         fsd = set(fsd, 'created', time());
0017 %         fsd = set(fsd, 'version', 'fsdata/set.m,v 1.7...');
0018 %
0019 % VERSION: $Id: set.m,v 1.13 2007/11/13 16:03:20 ingo Exp $
0020 %
0021 % HISTORY: 30-01-07 M Hewitson
0022 %           Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 VERSION = '$Id: set.m,v 1.13 2007/11/13 16:03:20 ingo Exp $';
0027
0028 % Check if this is a call for parameters
0029 if nargin == 2
0030     if isa(fsd, 'fsdata') && ischar(varargin{1})
0031         in = char(varargin{1});
0032         if strcmp(in, 'Params')
0033             fsd = plist;
0034             return
0035         elseif strcmp(in, 'Version')
0036             fsd = VERSION;
0037             return
0038         end
0039     end
0040 end
0041
0042 if (nargout ~= 0)
0043
0044     propArgin = varargin;
0045     while length(propArgin) >= 2
0046         prop = propArgin{1};
0047         val = propArgin{2};
0048         propArgin = propArgin(3:end);
0049         switch prop

```

```

0050     case 'name'
0051         fsd.name = val;
0052     case 'version'
0053         fsd.version = val;
0054     case 'created'
0055         fsd.created = val;
0056     case 'x'
0057         fsd.x = val;
0058     case 'y'
0059         fsd.y = val;
0060     case 'fs'
0061         fsd.fs = val;
0062     case 'enbw'
0063         fsd.enbw = val;
0064     case 'navs'
0065         fsd.navs = val;
0066     case 'xunits'
0067         fsd.xunits = val;
0068     case 'yunits'
0069         fsd.yunits = val;
0070     otherwise
0071         error([prop, ' is not a property of fsdata class']);
0072     end
0073 end
0074
0075 else
0076     if ischar(varargin{2})
0077         error('### please use: %s = set(%s, '%s'', '%s'');', ...
0078             inputname(1), ...
0079             inputname(1), ...
0080             varargin{1}, ...
0081             varargin{2});
0082     elseif isnumeric(varargin{2})
0083         error('### please use: %s = set(%s, '%s'', %d);', ...
0084             inputname(1), ...
0085             inputname(1), ...
0086             varargin{1}, ...
0087             varargin{2});
0088     else
0089         error('### please use: %s = set(%s, 'key'', 'value'');', ...
0090             inputname(1), ...
0091             inputname(1));
0092     end
0093 end

```

[Home](#) > [classes](#) > [@tsdata](#) > xmladd.m

xmladd

PURPOSE

XMLADD add a tsdata object to the given node of the input XML DOM

SYNOPSIS

```
function xml = xmladd(ds, xml, nodename, node)
```

DESCRIPTION

XMLADD add a tsdata object to the given node of the input XML DOM object.

M Hewitson 14-02-07

\$Id: xmladd.m,v 1.4 2007/11/09 14:07:39 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [get](#) GET get tsdata properties.
- [xml](#) XML convert a tsdata object to an DOM XML object.
- [xmladd](#) XMLADD add a tsdata object to the given node of the input XML DOM

This function is called by:

- [xml](#) XML convert a tsdata object to an DOM XML object.
- [xmladd](#) XMLADD add a tsdata object to the given node of the input XML DOM

SOURCE CODE

```
0001 function xml = xmladd(ds, xml, nodename, node)
0002
0003 % XMLADD add a tsdata object to the given node of the input XML DOM
0004 % object.
0005 %
0006 % M Hewitson 14-02-07
0007 %
0008 % $Id: xmladd.m,v 1.4 2007/11/09 14:07:39 ingo Exp $
```



```

0009 %
0010
0011 el = xml.docNode.createElement(nodename);
0012
0013 for j=1:length(ds)
0014
0015     d = ds(j);
0016
0017     elc = xml.docNode.createElement('tsdata');
0018
0019     % add name
0020     elcc = xml.docNode.createElement('name');
0021     elcc.appendChild(xml.docNode.createTextNode(d.name));
0022     elc.appendChild(elcc);
0023
0024     % add Fs
0025     elcc = xml.docNode.createElement('fs');
0026     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(d.fs)));
0027     elc.appendChild(elcc);
0028
0029     % add Xunits
0030     elcc = xml.docNode.createElement('xunits');
0031     elcc.appendChild(xml.docNode.createTextNode(d.xunits));
0032     elc.appendChild(elcc);
0033
0034     % add Yunits
0035     elcc = xml.docNode.createElement('yunits');
0036     elcc.appendChild(xml.docNode.createTextNode(d.yunits));
0037     elc.appendChild(elcc);
0038
0039     % add nsecs
0040     elcc = xml.docNode.createElement('nsecs');
0041     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(d.nsecs)));
0042     elc.appendChild(elcc);
0043
0044     % add t0
0045     xml = xmladd(get(d, 't0'), xml, 't0', elc);
0046
0047     % add Version
0048     elcc = xml.docNode.createElement('version');
0049     elcc.appendChild(xml.docNode.createTextNode(d.version));
0050     elc.appendChild(elcc);
0051
0052     % add Created
0053     xml = xmladd(get(d, 'created'), xml, 'created', elc);
0054
0055     if isreal(d.y)
0056
0057         % add data type
0058         elcc = xml.docNode.createElement('dtype');
0059         elcc.appendChild(xml.docNode.createTextNode('real'));

```

```

0060     elc.appendChild(elcc);
0061
0062     % add t,x
0063     elcc = xml.docNode.createElement('data');
0064     x = d.y;
0065     nmin = getappdata(0, 'xmlsetsize');
0066     while ~isempty(x)
0067         elccc = xml.docNode.createElement('yset');
0068         n = min(length(x), nmin);
0069         elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(x(1:n))));
0070 %         elccc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(x(1:n))));
0071         x = x(n+1:end);
0072         elcc.appendChild(elccc);
0073     end
0074     % add t
0075     x = d.x;
0076     nmin = getappdata(0, 'xmlsetsize');
0077     while ~isempty(x)
0078         elccc = xml.docNode.createElement('xset');
0079         n = min(length(x), nmin);
0080         elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(x(1:n))));
0081 %         elccc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(x(1:n))));
0082         x = x(n+1:end);
0083         elcc.appendChild(elccc);
0084     end
0085     elc.appendChild(elcc);
0086 else
0087     % add data type
0088     elcc = xml.docNode.createElement('dtype');
0089     elcc.appendChild(xml.docNode.createTextNode('complex'));
0090     elc.appendChild(elcc);
0091
0092     % add y
0093     elcc = xml.docNode.createElement('data');
0094     x = d.y;
0095     nmin = getappdata(0, 'xmlsetsize');
0096     while ~isempty(x)
0097         n = min(length(x), nmin);
0098         elccc = xml.docNode.createElement('ryset');
0099         elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(real(x(1:
n))))));
0100         elcc.appendChild(elccc);
0101         elccc = xml.docNode.createElement('iyset');
0102         elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(imag(x(1:
n))))));
0103         elcc.appendChild(elccc);
0104         x = x(n+1:end);
0105     end
0106     % add t
0107     x = d.x;
0108     nmin = getappdata(0, 'xmlsetsize');

```

```
0109     while ~isempty(x)
0110         elccc = xml.docNode.createElement('xset');
0111         n = min(length(x), nmin);
0112         elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(x(1:n))));
0113         x = x(n+1:end);
0114         elcc.appendChild(elccc);
0115     end
0116     elc.appendChild(elcc);
0117 end
0118 el.appendChild(elc);
0119 end
0120
0121 node.appendChild(el);
0122
0123
0124 % END
```

Generated on Tue 22-Jan-2008 10:39:13 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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(c1,c2, 'ao/name')

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 1.4 2007/12/04 18:16:11 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:

- **eq** EQ overloads the == operator for history objects.
- **ne** NE overloads the ~= operator for history objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002 % EQ overloads the == operator for history objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EQ overloads the == operator for history objects.
0007 %
0008 %           All fields are checked.
0009 %
0010 % CALL:      result = eq(c1,c2)
0011 %           result = eq(c1,c2,  exc_list)
0012 %           result = eq(c1,c2, 'property1', 'property2')
0013 %           result = eq(c1,c2, 'class/property', 'class/property')
0014 %
0015 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0016 %           result = eq(c1,c2, 'ao/name')
0017 %
0018 % INPUTS:    c1,c2      - input history objects
0019 %           exc_list - exception list
0020 %           List of properties which are not checked.
0021 %
0022 % OUTPUTS:   If the two history objects are considered equal, result == 1,
0023 %           otherwise, result == 0.
0024 %
0025 % VERSION:   $Id: eq.m,v 1.4 2007/12/04 18:16:11 ingo Exp $
0026 %
0027 % HISTORY:   29-08-2007 M Hewitson
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION = '$Id: eq.m,v 1.4 2007/12/04 18:16:11 ingo Exp $';
0033
0034 % Check if this is a call for parameters
0035 if nargin == 2
0036     if isa(c1, 'history') && ischar(c2)
0037         in = char(c2);
0038         if strcmp(in, 'Params')
0039             result = plist;
0040             return
0041         elseif strcmp(in, 'Version')

```

```

0042     result = VERSION;
0043     return
0044 end
0045 end
0046 end
0047
0048 result = 1;
0049
0050 %% Check class
0051 if ~strcmp(class(c1), class(c2))
0052     result = 0;
0053     return
0054 end
0055
0056 fields = fieldnames(c1);
0057
0058 for ii = 1:length(fields)
0059     field    = fields{ii};
0060
0061     ck_field = {field, ['history/' field]};
0062
0063     % Is the field equal to history then add the name 'history'
0064     % to the exception list.
0065     if strcmp(field, 'inhists')
0066         ck_field{end+1} = 'history';
0067         ck_field{end+1} = 'history/history';
0068     end
0069
0070     %% Check tag
0071     if ~(any(ismember(ck_field, varargin)))
0072
0073         if isobject(c1.(field))
0074
0075             if length(c1.(field)) ~= length(c2.(field))
0076                 result = 0;
0077                 return
0078             end
0079
0080             for jj = 1:length(c1.(field))
0081                 if ~eq(c1.(field)(jj), c2.(field)(jj), varargin{:})
0082                     result = 0;
0083                     disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0084                     return
0085                 end
0086             end
0087         else
0088             if ~isequalwithhequalnans(c1.(field), c2.(field))
0089                 result = 0;
0090                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0091                 return
0092             end
0093         end
0094

```

```

0095     end
0096 end
0097
0098
0099 % function result = eq(w1,w2, varargin)
0100 %
0101 % % EQ overloads the == operator for history objects.
0102 % %
0103 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
0104 % %
0105 % % DESCRIPTION: EQ overloads the == operator for history objects.
0106 % %
0107 % %           The 'name', 'version', 'plist', 'inhists', 'invars', 'created',
0108 % %           fields are checked.
0109 % %
0110 % % CALL:           result = eq(h1,h2)
0111 % %
0112 % % INPUTS:         h1,h2 - input specwin objects
0113 % %
0114 % % OUTPUTS:        If the two history objects are considered equal, result == 1,
0115 % %                  otherwise, result == 0.
0116 % %
0117 % % VERSION:        $Id: eq.m,v 1.4 2007/12/04 18:16:11 ingo Exp $
0118 % %
0119 % % HISTORY:         29-08-2007 M Hewitson
0120 % %                  Creation
0121 % %
0122 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
0123 %
0124 % % Assume equality to begin with
0125 % result = 1;
0126 %
0127 % %% Check name
0128 % if ~ismember('name', varargin)
0129 %     if ~strcmp(w1.name, w2.name)
0130 %         result = 0;
0131 %         return
0132 %     end
0133 % end
0134 %
0135 % %% Check version
0136 % if ~ismember('version', varargin)
0137 %     if ~strcmp(w1.version, w2.version)
0138 %         result = 0;
0139 %         return
0140 %     end
0141 % end
0142 %
0143 % %% Check plist
0144 % if ~ismember('plist', varargin)
0145 %     if xor(isempty(w1.plist), isempty(w2.plist))

```

```
0146 %     result = 0;
0147 % end
0148 % if ~eq(w1.plist, w2.plist, varargin{:})
0149 %     result = 0;
0150 %     return;
0151 % end
0152 % end
0153 %
0154 % %% check inhists
0155 %
0156 % if ~ismember('inhists', varargin)
0157 %     if length(w1.inhists) ~= length(w2.inhists)
0158 %         result = 0;
0159 %         return
0160 %     end
0161 %
0162 %     for j=1:length(w1.inhists)
0163 %         if ~eq(w1.inhists(j), w2.inhists(j), varargin{:})
0164 %             result = 0;
0165 %             return
0166 %         end
0167 %     end
0168 % end
0169 %
0170 % %% check invars
0171 % if ~ismember('invars', varargin)
0172 %     if length(w1.invars) ~= length(w2.invars)
0173 %         result = 0;
0174 %         return
0175 %     end
0176 %
0177 %     for j=1:length(w1.invars)
0178 %         if ~strcmp(w1.invars{j}, w2.invars{j})
0179 %             result = 0;
0180 %             return
0181 %         end
0182 %     end
0183 % end
0184 %
0185 % %% check created
0186 % if ~ismember('created', varargin)
0187 %     if w1.created ~= w2.created
0188 %         result = 0;
0189 %         return
0190 %     end
0191 % end
```


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)

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 1.9 2007/11/14 10:11:40 hewitson Exp \$

HISTORY: 30-08-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [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.

This function is called by:

SOURCE CODE

```

0001 function varargout = ltpda_obj_retrieve(varargin)
0002
0003 % LTPDA_OBJ_RETRIEVE retrieves a collection of objects from an LTPDA repository.
0004 %
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: RETRIEVE a collection of objects from an LTPDA repository.
0009 %
0010 % CALL:      objs      = ltpda_obj_retrieve(conn, obj_id_1, obj_id_2)
0011 %            [o1,o2] = ltpda_obj_retrieve(conn, obj_id_1, obj_id_2)
0012 %            objs      = ltpda_obj_retrieve(conn, 'Collection', coll_id)
0013 %
0014 % INPUTS:
0015 %           conn        - database connection object
0016 %           obj_id_N    - an object ID
0017 %           coll_id     - a collection ID
0018 %
0019 % OUTPUTS:
0020 %           objs        - the retrieved object(s) as a cell array.*
0021 %           o1,o2,...,oN - returns the first N objects
0022 %
0023 %
0024 % If more than one object is retrieved and only one output is specified
0025 % then the output is a cell array of objects.
0026 %
0027 % If only a single object is requested, it is returned as an object,
0028 % not packed in a cell array.
0029 %
0030 % VERSION:      $Id: ltpda_obj_retrieve.m,v 1.9 2007/11/14 10:11:40 hewitson Exp
0031 %
0032 % HISTORY: 30-08-07 M Hewitson
0033 %           Creation
0034 %
0035 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0036
0037 objs = [];
0038 conn = varargin{1};

```

```

0039 if ~isa(conn, 'database')
0040     error('### the first argument should be a database connection object.');
```

```

0041 end
0042
0043 if nargin == 3 && ischar(varargin{2}) && isnumeric(varargin{3})
0044     if strcmp(varargin{2}, 'Collection')
0045         cid = varargin{3};
0046         % Get a list of object IDs from the collection ID
0047         ids = mysql_getObjIds(conn, cid);
0048     else
0049         help(mfilename)
0050         error('### Incorrect usage');
```

```

0051     end
0052 elseif nargin > 1 && isnumeric(varargin{2})
0053     ids = [varargin{2:end}];
0054 else
0055     error('### Incorrect usage');
```

```

0056 end
0057
0058 disp(['*** retrieving objects [' num2str(ids) ']'])
0059
0060 for j=1:length(ids)
0061     tt = mysql_getObjType(conn, ids(j));
0062     cmd = sprintf('obj = %s(conn, ids(j));', tt);
0063     eval(cmd);
0064
0065     % retrieve the hash for this object
0066     % md5hash = mysql_getMD5hash(conn, ids(j));
0067
0068     % compute hash of object
0069     % x      = xml(obj);
0070     % otxt = xmlwrite(x.docNode);
0071     % h      = ltpda_hash(otxt, 'MD5');
```

```

0072
0073     % if ~strcmp(h, md5hash)
0074     %     disp(['! compared: ' md5hash ' - ' h])
0075     %     error('### MD5 check failed for object %d.', ids(j));
0076     % else
0077     if j==1
0078         objs = {obj};
0079     else
0080         objs = [objs {obj}];
0081     end
0082     % end
0083 end
0084
0085 if nargout == 1
0086     if length(objs) == 1
0087         varargout{1} = objs{1};
0088     else
0089         varargout{1} = objs;
0090     end
0091 else

```

```
0092     for k=1:nargout
0093         varargout{k} = objs{k};
0094     end
0095 end
0096 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@history](#) > set.m

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 1.10 2007/12/05 16:26:22 ingo Exp \$

HISTORY: 30-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a param object into a string.

This function is called by:

- [getNode](#) GETNODES converts a history object to a nodes structure suitable for plotting as a tree.
- [getNode_plot](#) GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a

tree.

- **plot** PLOT plots a history object as a tree diagram.

SOURCE CODE

```

0001 function h = set(h, varargin)
0002 % SET set a history property.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET set a history property.
0007 %
0008 % CALL: h = set(h, 'name',    'name');
0009 %       h = set(h, 'inhist',  history);
0010 %       h = set(h, 'invars',  {});
0011 %       h = set(h, 'plist',   plist);
0012 %       h = set(h, 'n',       n);
0013 %       h = set(h, 'pn',      pn);
0014 %       h = set(h, 'created',  time());
0015 %       h = set(h, 'version',  'history/set.m,v 1.7...');
0016 %       h = set(h, 'consver',  'history/set.m,v 1.7...');
0017 %
0018 % VERSION: $Id: set.m,v 1.10 2007/12/05 16:26:22 ingo Exp $
0019 %
0020 % HISTORY: 30-01-07 M Hewitson
0021 %           Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: set.m,v 1.10 2007/12/05 16:26:22 ingo Exp $';
0026
0027 % Check if this is a call for parameters
0028 if nargin == 2
0029     if isa(h, 'history') && ischar(varargin{1})
0030         in = char(varargin{1});
0031         if strcmp(in, 'Params')
0032             h = plist;
0033             return
0034         elseif strcmp(in, 'Version')
0035             h = VERSION;
0036             return
0037         end
0038     end
0039 end
0040
0041 if (nargout ~= 0)
0042
0043     propArgin = varargin;
0044
0045     while length(propArgin) >= 2
0046         prop = varargin{1};

```

```

0047     val = varargin{2};
0048     propArgin = propArgin(3:end);
0049     switch prop
0050     case 'name'
0051         h.name = val;
0052     case 'created'
0053         h.created = val;
0054     case 'version'
0055         h.version = val;
0056     case 'plist'
0057         h.plist = val;
0058     case 'n'
0059         h.n = val;
0060     case 'pn'
0061         h.pn = val;
0062     case 'inhists'
0063         h.inhists = val;
0064     case 'invars'
0065         h.invars = val;
0066     case 'consver'
0067         h.consver = val;
0068     otherwise
0069         error([val, ' is not a property of history class']);
0070     end
0071 end
0072
0073 else
0074     if ischar(varargin{2})
0075         error('### please use: %s = set(%s, '%s'', '%s'');', ...
0076             inputname(1), ...
0077             inputname(1), ...
0078             varargin{1}, ...
0079             varargin{2});
0080     elseif isnumeric(varargin{2})
0081         error('### please use: %s = set(%s, '%s'', %d);', ...
0082             inputname(1), ...
0083             inputname(1), ...
0084             varargin{1}, ...
0085             varargin{2});
0086     else
0087         error('### please use: %s = set(%s, key, 'value'');', ...
0088             inputname(1), ...
0089             inputname(1));
0090     end
0091 end

```

[Home](#) > [classes](#) > [@xydata](#) > xmladd.m

xmladd

PURPOSE

XMLADD add a xydata object to the given node of the input XML DOM

SYNOPSIS

function xml = xmladd(ds, xml, nodename, node)

DESCRIPTION

XMLADD add a xydata object to the given node of the input XML DOM object.

M Hewitson 14-02-07

\$Id: xmladd.m,v 1.3 2007/10/18 10:09:26 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [get](#) GET gets xydata properties.
- [xml](#) XML convert an xydata object to an DOM XML object.
- [xmladd](#) XMLADD add a xydata object to the given node of the input XML DOM

This function is called by:

- [xml](#) XML convert an xydata object to an DOM XML object.
- [xmladd](#) XMLADD add a xydata object to the given node of the input XML DOM

SOURCE CODE

```
0001 function xml = xmladd(ds, xml, nodename, node)
0002
0003 % XMLADD add a xydata object to the given node of the input XML DOM
0004 % object.
0005 %
0006 % M Hewitson 14-02-07
0007 %
```



```

0008 % $Id: xmladd.m,v 1.3 2007/10/18 10:09:26 hewitson Exp $
0009 %
0010
0011 el = xml.docNode.createElement(nodename);
0012
0013 for j=1:length(ds)
0014
0015     d = ds(j);
0016
0017     elc = xml.docNode.createElement('xydata');
0018
0019     % add name
0020     elcc = xml.docNode.createElement('name');
0021     elcc.appendChild(xml.docNode.createTextNode(d.name));
0022     elc.appendChild(elcc);
0023
0024     % add Xunits
0025     elcc = xml.docNode.createElement('xunits');
0026     elcc.appendChild(xml.docNode.createTextNode(d.xunits));
0027     elc.appendChild(elcc);
0028
0029     % add Yunits
0030     elcc = xml.docNode.createElement('yunits');
0031     elcc.appendChild(xml.docNode.createTextNode(d.yunits));
0032     elc.appendChild(elcc);
0033
0034     % add Version
0035     elcc = xml.docNode.createElement('version');
0036     elcc.appendChild(xml.docNode.createTextNode(d.version));
0037     elc.appendChild(elcc);
0038
0039     % add Created
0040     xml = xmladd(get(d, 'created'), xml, 'created', elc);
0041
0042     if isreal(d.y)
0043         % add data type
0044         elcc = xml.docNode.createElement('dtype');
0045         elcc.appendChild(xml.docNode.createTextNode('real'));
0046         elc.appendChild(elcc);
0047
0048         % add real x
0049         elcc = xml.docNode.createElement('data');
0050         x = d.x;
0051         nmin = getappdata(0, 'xmlsetsize');
0052         while ~isempty(x)
0053             elccc = xml.docNode.createElement('xset');
0054             n = min(length(x), nmin);
0055             elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(x(1:n))));
0056             x = x(n+1:end);
0057             elcc.appendChild(elccc);

```

```

0058     end
0059     % add y
0060     y      = d.y;
0061     nmin = getappdata(0, 'xmlsetsize');
0062     while ~isempty(y)
0063         elccc = xml.docNode.createElement('yset');
0064         n = min(length(y), nmin);
0065         elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(y(1:n))));
0066         y = y(n+1:end);
0067         elcc.appendChild(elccc);
0068     end
0069     elc.appendChild(elcc);
0070 else
0071     % add data type
0072     elcc = xml.docNode.createElement('dtype');
0073     elcc.appendChild(xml.docNode.createTextNode('complex'));
0074     elc.appendChild(elcc);
0075
0076     % add complex Y
0077     elcc = xml.docNode.createElement('data');
0078     y      = d.y;
0079     nmin = getappdata(0, 'xmlsetsize');
0080     while ~isempty(y)
0081         n = min(length(y), nmin);
0082         elccc = xml.docNode.createElement('ryset');
0083         elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(real(y(1:
n))))));
0084         elcc.appendChild(elccc);
0085         elccc = xml.docNode.createElement('iyset');
0086         elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(imag(y(1:
n))))));
0087         elcc.appendChild(elccc);
0088         y = y(n+1:end);
0089     end
0090     % add X
0091     x      = d.x;
0092     nmin = getappdata(0, 'xmlsetsize');
0093     while ~isempty(x)
0094         elccc = xml.docNode.createElement('xset');
0095         n = min(length(x), nmin);
0096         elccc.appendChild(xml.docNode.createTextNode(ltpda_num2str(x(1:n))));
0097         x = x(n+1:end);
0098         elcc.appendChild(elccc);
0099     end
0100     elc.appendChild(elcc);
0101 end
0102
0103 el.appendChild(elc);
0104 end
0105

```

```
0106 node.appendChild(el);  
0107  
0108  
0109 % END
```

Generated on Tue 22-Jan-2008 10:39:13 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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(c1,c2, 'ao/name')

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 1.3 2007/12/06 17:09:15 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:

- **eq** EQ overloads the == operator for mfir objects.
- **ne** NE overloads the ~= operator for mfir objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002 % EQ overloads the == operator for mfir objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EQ overloads the == operator for mfir objects.
0007 %
0008 %           All fields are checked.
0009 %
0010 % CALL:      result = eq(c1,c2)
0011 %           result = eq(c1,c2, exc_list)
0012 %           result = eq(c1,c2, 'property1', 'property2')
0013 %           result = eq(c1,c2, 'class/property', 'class/property')
0014 %
0015 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0016 %           result = eq(c1,c2, 'ao/name')
0017 %
0018 % INPUTS:    c1,c2      - input mfir objects
0019 %           exc_list - exception list
0020 %           List of properties which are not checked.
0021 %
0022 % OUTPUTS:   If the two mfir objects are considered equal, result == 1,
0023 %           otherwise, result == 0.
0024 %
0025 % VERSION:   $Id: eq.m,v 1.3 2007/12/06 17:09:15 ingo Exp $
0026 %
0027 % HISTORY:   29-08-2007 M Hewitson
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION = '$Id: eq.m,v 1.3 2007/12/06 17:09:15 ingo Exp $';
0033
0034 % Check if this is a call for parameters
0035 if nargin == 2
0036     if isa(c1, 'mfir') && ischar(c2)
0037         in = char(c2);
0038         if strcmp(in, 'Params')
0039             result = plist;
0040             return

```

```
0041     elseif strcmp(in, 'Version')
0042         result = VERSION;
0043         return
0044     end
0045 end
0046 end
0047
0048 result = 1;
0049
0050 %% Check class
0051 if ~strcmp(class(c1), class(c2))
0052     result = 0;
0053     return
0054 end
0055
0056 fields = fieldnames(c1);
0057
0058 for ii = 1:length(fields)
0059     field = fields{ii};
0060
0061     ck_field = {field, ['mfir/' field]};
0062
0063     %% Check tag
0064     if ~(any(ismember(ck_field, varargin)))
0065
0066         if isobject(c1.(field))
0067             if ~eq(c1.(field), c2.(field), varargin{:})
0068                 result = 0;
0069                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0070                 return
0071             end
0072         else
0073             if ~isequalwithequalnans(c1.(field), c2.(field))
0074                 result = 0;
0075                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0076                 return
0077             end
0078         end
0079     end
0080 end
0081 end
0082
```

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 1.13 2007/11/13 13:53:47 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 id = getObjID\(conn, hash\)](#)
- [function Cid = getCollectionID\(conn, objids\)](#)
- [function db_lock\(conn\)](#)
- [function db_unlock\(conn\)](#)
- [function s = csv\(x\)](#)

SOURCE CODE

```

0001 function varargout = ltpda_obj_submit(varargin)
0002
0003 % LTPDA_OBJ_SUBMIT submits the given collection of objects to an LTPDA
Repository.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: Submits the given collection of objects to an LTPDA Repository.
0008 %               The input object(s) will be submitted to the LTPDA Repository
0009 %               configured in ltpda_startup.m. If multiple objects are submitted
0010 %               together, a corresponding collection entry will be made.
0011 %
0012 %               This is meant to be called by class/submit methods.
0013 %
0014 % CALL:         [ids, cids] = ltpda_obj_submit(o1, sinfo)
0015 %               [ids, cids] = ltpda_obj_submit(o1,o2, sinfo)
0016 %               [ids, cids] = ltpda_obj_submit([o1 o2], o3, {o4, o5}, sinfo)
0017 %
0018 % INPUTS:
0019 %               o1,o2,... - input objects to be submitted
0020 %               sinfo      - a structure containing the following fields:
0021 %
0022 %               'conn'      - database connection object
0023 %               'username'   - the username for the server
0024 %               'experiment_title' - a title for the submission
0025 %               'experiment_description' - a description of this submission
0026 %               'reference_ids' - a string containing any
0027 %                               reference object id numbers
0028 %               'additional_comments' - any additional comments
0029 %               'additional_authors' - any additional author names

```



```

0030 %
0031 % VERSION:      $Id: ltpda_obj_submit.m,v 1.13 2007/11/13 13:53:47 hewitson Exp $
0032 %
0033 %
0034 % HISTORY: 09-05-07 M Hewitson
0035 %           Creation
0036 %
0037 %%%%%%%%%%%
0038
0039 %% Process inputs
0040
0041 objs = {};
0042 sinfo = [];
0043 for j=1:nargin
0044     if ltpda_isobject(varargin{j})
0045         ins = varargin{j};
0046         if iscell(ins)
0047             for k=1:length(ins)
0048                 objs = [objs {ins{k}}];
0049             end
0050         else
0051             for k=1:length(ins)
0052                 objs = [objs {ins(k)}];
0053             end
0054         end
0055     elseif isstruct(varargin{j})
0056         sinfo = varargin{j};
0057     else
0058         warning(sprintf('!!! input argument %d will be ignored.', j));
0059     end
0060 end
0061
0062 if isempty(objs)
0063     error('### Please input at least one LTPDA object.');
```

```

0064 end
0065 if isempty(sinfo)
0066     error('### Please provide an input submission structure.');
```

```

0067 end
0068
0069
0070 %% Some setup
0071
0072 % make copy of connection object
0073 conn = sinfo.conn;
0074 if isempty(conn)
0075     error('### No valid database connect supplied. Aborting submit process.');
```

```

0076 end
0077
0078
0079 % Machine details
0080 p = provenance;
0081
0082 % comment mapping

```

```

0083 comm1 = sinfo.experiment_title;
0084 comm2 = sinfo.experiment_description;
0085 comm3 = sinfo.reference_ids;
0086 comm4 = sinfo.additional_comments;
0087 comm5 = sinfo.additional_authors;
0088 comm6 = '';
0089
0090 %% Process each object and collect id numbers
0091
0092 disp('*** Submitting objects to repository...');
0093
0094 try
0095
0096     % Look-up user id
0097     userid = mysql_getUserID(conn, sinfo.username);
0098     disp(sprintf(' ** got user id %d for user: %s', userid, sinfo.username));
0099     if userid < 1 || isnan(userid)
0100         error('### Unknown username.');
```

```

0101     end
0102
0103     %%%%%%%%% Lock database
0104     % db_lock(conn);
0105     % disp(' ** database locked for submission...');
```

```

0106
0107     % The date for the transaction table
0108     t      = time();
0109     tdate = t.time_str;
0110
0111     ids = [];
0112     for o=1:length(objs)
0113         % this object
0114         obj = objs{o};
0115
0116         % Check this is a user object that can be submitted
0117         if isa(obj, 'ao') || isa(obj, 'miir') || isa(obj, 'mfir') || ...
0118             isa(obj, 'pzmodel') || isa(obj, 'time') || isa(obj, 'timespan') || ...
0119             isa(obj, 'specwin') || isa(obj, 'plist')
0120
0121
0122             %%%%%%%%% Some setup
0123
0124             % Object name
0125             if isfield(obj, 'name')
0126                 name = obj.name;
0127             else
0128                 name = '';
0129             end
0130
0131             % Data object (if present)
0132             if isfield(obj, 'data') && ~isempty(obj.data)
0133                 dname = obj.data.name;
0134                 dtype = class(obj.data);
0135                 ndata = length(get_xy_values(obj.data));
```

```

0136     else
0137         dtype = '';
0138         dname = '';
0139         ndata = 0;
0140     end
0141
0142     % Creation time
0143     if isfield(obj, 'created')
0144         if isa(obj.created, 'time')
0145             created = obj.created.time_str;
0146         elseif ischar(obj.created)
0147             created = obj.created;
0148         else
0149             error('### Unknown format for created field.');
```

0150 end

0151 else

0152 created = '0000-00-00 00:00:00';

0153 end

0154 if isempty(created)

0155 created = '0000-00-00 00:00:00';

0156 end

0157

0158 % Version

0159 if isfield(obj, 'version')

0160 objver = obj.version;

0161 else

0162 objver = '';

0163 end

0164

0165 %%%%%%%%%%% Convert object to XML

0166 x = xml(obj); % first to DOM

0167 otxt = (xmlwrite(x.docNode)); % then to string

0168

0169 %%%%%%%%%%% Create MD5 hash of object

0170 md5hash = ltpda_hash(otxt, 'MD5');

0171

0172 %%%%%%%%%%% Submit object to objs table

0173

0174 curs = exec(conn, ['insert into objs(xml,hash) values('' otxt '', ''

char(md5hash) '');']);

0175 close(curs);

0176

0177 objid = getObjID(conn, md5hash);

0178 disp(sprintf(' + submitted object %s with id %d', class(obj), objid));

0179

0180 %%%%%%%%%%% Make objmeta entry

0181

0182 message = ltpda_insert(conn, 'objmeta', ...

0183 'obj_id', objid, ...

0184 'obj_type', class(obj), ...

0185 'name', name, ...

0186 'created', created, ...

0187 'version', objver, ...

```

0188         'ip', p.ip, ...
0189         'hostname', p.hostname, ...
0190         'os', p.os, ...
0191         'submitted', datestr(now, 31), ...
0192         'comment1', comm1, ...
0193         'comment2', comm2, ...
0194         'comment3', comm3, ...
0195         'comment4', comm4, ...
0196         'comment5', comm5, ...
0197         'comment6', comm6...
0198     );
0199     disp(sprintf(' + made meta-data entry'));
0200     %         'filename', sprintf('OBJ%06d.xml', objid)...
0201
0202     %%%%%%%%%%% Update transactions table
0203
0204     message = ltpda_insert(conn, 'transactions', ...
0205         'obj_id', objid, ...
0206         'user_id', userid, ...
0207         'transdate', tdate, ...
0208         'direction', 'in'...
0209     );
0210     disp(sprintf(' + updated transactions table'));
0211
0212     % Collect this ID
0213     ids = [ids objid];
0214 else
0215     warning('!! Object is not one of the User classes. Not submitting.');
```

disp(obj);

```

0217 end
0218 end
0219
0220 %% Now make collection entry
0221 if ~isempty(ids)
0222     message = ltpda_insert(conn, 'collections', ...
0223         'nobjs', length(ids), ...
0224         'obj_ids', csv(ids)...
0225     );
0226     disp(sprintf(' ** made collection entry'));
0227 end
0228 catch
0229     disp('### Submission error - cleaning up.');
```

0230

```

0231 % % unlock database
0232 % db_unlock(conn);
0233
0234 rethrow(lasterror)
0235 end
0236
0237 %% unlock database
0238 % disp(sprintf(' ** unlocking tables.');
```

0239 % db_unlock(conn);

```

0240
0241 varargout{1} = ids;
0242 varargout{2} = getCollectionID(conn, ids);
0243
0244 disp('*** submission complete.');
```

```

0245
0246 end
0247
0248 %-----
0249 %
0250 function id = getObjID(conn, hash)
0251
0252 q = sprintf('select last_insert_id()');
0253 curs = exec(conn, q);
0254 curs = fetch(curs);
0255 id = curs.Data{1};
0256 close(curs);
0257
0258 end
0259
0260 %-----
0261 %
0262 function Cid = getCollectionID(conn, objids)
0263
0264 curs = exec(conn, sprintf('select id from collections where obj_ids="%s"', csv
(objids)));
0265 curs = fetch(curs);
0266 Cid = curs.Data{1};
0267 close(curs);
0268
0269 if strcmp(Cid, 'No Data')
0270     Cid = [];
0271 end
0272
0273 end
0274
0275 %-----
0276 %
0277 function db_lock(conn)
0278
0279 % lock all tables
0280 q = ['lock tables '...
0281     'ao write,'...
0282     'collections write,'...
0283     'fsdata write,'...
0284     'mfir write,'...
0285     'miir write,'...
0286     'objmeta write,'...
0287     'objs write,'...
0288     'transactions write,'...
0289     'tsdata write,'...
0290     'users write'];
```

```
0291 curs = exec(conn, q);
0292
0293 end
0294
0295
0296 %-----
0297 %
0298 function db_unlock(conn)
0299
0300 q = 'UNLOCK TABLES';
0301 curs = exec(conn, q);
0302
0303 end
0304
0305 %-----
0306 % make comma separated list of numbers
0307 function s = csv(x)
0308
0309 s = sprintf('%g, ', x);
0310 s = s(1:end-1);
0311
0312 end
0313
0314 % END
```

[Home](#) > [classes](#) > [@mfir](#) > set.m

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 1.5 2007/12/10 18:07:17 ingo Exp \$

HISTORY: 30-01-2007 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a mfir object into a string.

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

```

0001 function f = set(f, varargin)
0002 % SET set a mfir object property.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET set an mfir object property.
0007 %
0008 % CALL:          f      = set(f, property, value);
0009 %               mfir = set(mfir, 'name',      'name');
0010 %               mfir = set(mfir, 'fs',        0);
0011 %               mfir = set(mfir, 'ntaps',     0);
0012 %               mfir = set(mfir, 'a',         []);
0013 %               mfir = set(mfir, 'gd',        0);
0014 %               mfir = set(mfir, 'gain',      0);
0015 %               mfir = set(mfir, 'histout',   history());
0016 %               mfir = set(mfir, 'infile',    'name');
0017 %               mfir = set(mfir, 'plist',     plist);
0018 %               mfir = set(mfir, 'created',   time());
0019 %               mfir = set(mfir, 'version',   'version');
0020 %
0021 % VERSION:       $Id: set.m,v 1.5 2007/12/10 18:07:17 ingo Exp $
0022 %
0023 % HISTORY:       30-01-2007 M Hewitson
0024 %               Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0027
0028 VERSION = '$Id: set.m,v 1.5 2007/12/10 18:07:17 ingo Exp $';
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(f, 'mfir') && ischar(varargin{1})
0033         in = char(varargin{1});
0034         if strcmp(in, 'Params')
0035             f = plist;
0036             return
0037         elseif strcmp(in, 'Version')
0038             f = VERSION;
0039             return
0040         end
0041     end
0042 end
0043
0044 propArgin = varargin;

```



```
0045 while length(propArgin) >= 2
0046     prop = propArgin{1};
0047     val = propArgin{2};
0048     propArgin = propArgin(3:end);
0049     switch prop
0050         case 'name'
0051             f.name = val;
0052         case 'version'
0053             f.version = val;
0054         case 'created'
0055             f.created = val;
0056         case 'fs'
0057             f.fs = val;
0058         case 'ntaps'
0059             f.ntaps = val;
0060         case 'a'
0061             f.a = val;
0062         case 'gd'
0063             f.gd = val;
0064         case 'gain'
0065             f.gain = val;
0066         case 'infile'
0067             f.infile = val;
0068         case 'histout'
0069             f.histout = val;
0070         case 'plist'
0071             f.plist = val;
0072         otherwise
0073             error([prop, ' is not a valid mfir property.']);
0074     end
0075 end
```

[Home](#) > [classes](#) > [@zero](#) > xmladd.m

xmladd

PURPOSE

XMLADD add a zero object to the given node of the input XML DOM

SYNOPSIS

```
function xml = xmladd(ps, xml, nodename, node)
```

DESCRIPTION

XMLADD add a zero object to the given node of the input XML DOM object.

M Hewitson 14-02-07

\$Id: xmladd.m,v 1.2 2007/10/16 17:37:21 ingo Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

- [get](#) GET get zero properties.
- [xml](#) XML convert a zero object to an DOM XML object.
- [xmladd](#) XMLADD add a zero object to the given node of the input XML DOM

This function is called by:

- [xml](#) XML convert a zero object to an DOM XML object.
- [xmladd](#) XMLADD add a zero object to the given node of the input XML DOM

SOURCE CODE

```
0001 function xml = xmladd(ps, xml, nodename, node)
0002
0003 % XMLADD add a zero object to the given node of the input XML DOM
0004 % object.
0005 %
0006 % M Hewitson 14-02-07
0007 %
```

```

0008 % $Id: xmladd.m,v 1.2 2007/10/16 17:37:21 ingo Exp $
0009 %
0010
0011 el = xml.docNode.createElement(nodename);
0012
0013 %     p.name     = 'real pole';
0014 %     p.f        = varargin{1};
0015 %     p.q        = -1;
0016 %     p.ri       = p.f;
0017 %     p.version  = VERSION;
0018
0019 for j=1:length(ps)
0020     p = ps(j);
0021
0022     % Add pole object
0023     elc = xml.docNode.createElement('zero');
0024
0025     % add name
0026     elcc = xml.docNode.createElement('name');
0027     elcc.appendChild(xml.docNode.createTextNode(get(p, 'name')));
0028     elc.appendChild(elcc);
0029
0030     % add version
0031     elcc = xml.docNode.createElement('version');
0032     elcc.appendChild(xml.docNode.createTextNode(get(p, 'version')));
0033     elc.appendChild(elcc);
0034
0035     % add Created
0036     xml = xmladd(p.created, xml, 'created', elc);
0037
0038     % add f
0039     elcc = xml.docNode.createElement('f');
0040     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(get(p, 'f'))));
0041     elc.appendChild(elcc);
0042
0043     % add Q
0044     elcc = xml.docNode.createElement('q');
0045     elcc.appendChild(xml.docNode.createTextNode(ltpda_mat2str(get(p, 'q'))));
0046     elc.appendChild(elcc);
0047
0048     el.appendChild(elc);
0049 end
0050
0051 node.appendChild(el);
0052
0053

```


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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(c1,c2, 'ao/name')

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 1.3 2007/12/06 17:09:15 ingo Exp \$

HISTORY: 29-08-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- **char** CHAR convert a miir object into a string.
- **eq** EQ overloads the == operator for miir objects.

This function is called by:

- **eq** EQ overloads the == operator for miir objects.
- **ne** NE overloads the ~= operator for miir objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002 % EQ overloads the == operator for miir objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EQ overloads the == operator for miir objects.
0007 %
0008 %           All fields are checked.
0009 %
0010 % CALL:      result = eq(c1,c2)
0011 %           result = eq(c1,c2, exc_list)
0012 %           result = eq(c1,c2, 'property1', 'property2')
0013 %           result = eq(c1,c2, 'class/property', 'class/property')
0014 %
0015 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0016 %           result = eq(c1,c2, 'ao/name')
0017 %
0018 % INPUTS:    c1,c2      - input miir objects
0019 %           exc_list - exception list
0020 %           List of properties which are not checked.
0021 %
0022 % OUTPUTS:   If the two miir objects are considered equal, result == 1,
0023 %           otherwise, result == 0.
0024 %
0025 % VERSION:   $Id: eq.m,v 1.3 2007/12/06 17:09:15 ingo Exp $
0026 %
0027 % HISTORY:   29-08-2007 M Hewitson
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION    = '$Id: eq.m,v 1.3 2007/12/06 17:09:15 ingo Exp $';
0033
0034 % Check if this is a call for parameters
0035 if nargin == 2
0036     if isa(c1, 'miir') && ischar(c2)
0037         in = char(c2);
0038         if strcmp(in, 'Params')
0039             result = plist;
0040             return

```

```
0041     elseif strcmp(in, 'Version')
0042         result = VERSION;
0043         return
0044     end
0045 end
0046 end
0047
0048 result = 1;
0049
0050 %% Check class
0051 if ~strcmp(class(c1), class(c2))
0052     result = 0;
0053     return
0054 end
0055
0056 fields = fieldnames(c1);
0057
0058 for ii = 1:length(fields)
0059     field = fields{ii};
0060
0061     ck_field = {field, ['miir/' field]};
0062
0063     %% Check tag
0064     if ~(any(ismember(ck_field, varargin)))
0065
0066         if isobject(c1.(field))
0067             if ~eq(c1.(field), c2.(field), varargin{:})
0068                 result = 0;
0069                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0070                 return
0071             end
0072         else
0073             if ~isequalwithequalnans(c1.(field), c2.(field))
0074                 result = 0;
0075                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0076                 return
0077             end
0078         end
0079     end
0080 end
0081 end
0082
```

[Home](#) > [m](#) > [helper](#) > ltpda_phase.m

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)
```



```

0002 % LTPDA_PHASE return the phase in degrees for a given complex input.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_PHASE return the phase in degrees for a given complex
0007 %               input. Supposed to be analogous to angle but return degrees
0008 %               instead of radians.
0009 %
0010 % CALL:         p = ltpda_phase(resp)
0011 %
0012 % INPUTS:       resp - complex number
0013 %
0014 % OUTPUTS:      p     - phase
0015 %
0016 % VERSION: $Id: ltpda_phase.m,v 1.5 2007/07/13 12:17:39 ingo Exp $
0017 %
0018 % HISTORY: dd-mm-yyyy M Hewitson
0019 %           Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 p = angle(resp)*180/pi;
0024
0025 % END

```

[Home](#) > [classes](#) > [@miir](#) > set.m

set

PURPOSE

SET set a miir object property.

SYNOPSIS

function f = set(f, varargin)

DESCRIPTION

SET set a miir object property.

%%%

DESCRIPTION: SET set a miir object property.

CALL:

```
miir = set(miir, property, value);
miir = set(miir, 'name', 'name');
miir = set(miir, 'fs', 0);
miir = set(miir, 'ntaps', 0);
miir = set(miir, 'a', []);
miir = set(miir, 'b', []);
miir = set(miir, 'gain', 0);
miir = set(miir, 'histin', history());
miir = set(miir, 'histout', history());
miir = set(miir, 'infile', 'name');
miir = set(miir, 'plist', plist());
miir = set(miir, 'created', time());
miir = set(miir, 'version', '$Date: 2007/12/10 18:07:17 $');
```

VERSION: \$Id: set.m,v 1.8 2007/12/10 18:07:17 ingo Exp \$

HISTORY: 30-01-2007 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a miir object into a string.

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

```

0001 function f = set(f, varargin)
0002 % SET set a miir object property.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET set a miir object property.
0007 %
0008 % CALL:      miir = set(miir, property, value);
0009 %            miir = set(miir, 'name', 'name');
0010 %            miir = set(miir, 'fs', 0);
0011 %            miir = set(miir, 'ntaps', 0);
0012 %            miir = set(miir, 'a', []);
0013 %            miir = set(miir, 'b', []);
0014 %            miir = set(miir, 'gain', 0);
0015 %            miir = set(miir, 'histin', history());
0016 %            miir = set(miir, 'histout', history());
0017 %            miir = set(miir, 'infile', 'name');
0018 %            miir = set(miir, 'plist', plist());
0019 %            miir = set(miir, 'created', time());
0020 %            miir = set(miir, 'version', '$Date: 2007/12/10 18:07:17 $');
0021 %
0022 % VERSION:    $Id: set.m,v 1.8 2007/12/10 18:07:17 ingo Exp $
0023 %
0024 % HISTORY:    30-01-2007 M Hewitson
0025 %              Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 VERSION = '$Id: set.m,v 1.8 2007/12/10 18:07:17 ingo Exp $';
0030
0031 % Check if this is a call for parameters
0032 if nargin == 2
0033     if isa(f, 'miir') && ischar(varargin{1})
0034         in = char(varargin{1});
0035         if strcmp(in, 'Params')
0036             f = plist;
0037             return
0038         elseif strcmp(in, 'Version')
0039             f = VERSION;
0040             return
0041         end
0042     end
0043 end

```

```
0044
0045 propArgin = varargin;
0046 while length(propArgin) >= 2
0047     prop = propArgin{1};
0048     val = propArgin{2};
0049     propArgin = propArgin(3:end);
0050     switch prop
0051         case 'name'
0052             f.name = val;
0053         case 'version'
0054             f.version = val;
0055         case 'created'
0056             f.created = val;
0057         case 'fs'
0058             f.fs = val;
0059         case 'ntaps'
0060             f.ntaps = val;
0061         case 'a'
0062             f.a = val;
0063         case 'b'
0064             f.b = val;
0065         case 'gain'
0066             f.gain = val;
0067         case 'infile'
0068             f.infile = val;
0069         case 'histin'
0070             f.histin = val;
0071         case 'histout'
0072             f.histout = val;
0073         case 'plist'
0074             f.plist = val;
0075         otherwise
0076             error([prop, ' is not a valid miir property.']);
0077     end
0078 end
```

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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(c1,c2, 'ao/name')

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 1.5 2007/12/11 17:27:09 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:

- **eq** EQ overloads the == operator for param objects.
- **ne** NE overloads the ~= operator for param objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002 % EQ overloads the == operator for param objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EQ overloads the == operator for param objects.
0007 %
0008 %           All fields are checked.
0009 %
0010 % CALL:      result = eq(c1,c2)
0011 %           result = eq(c1,c2, exc_list)
0012 %           result = eq(c1,c2, 'property1', 'property2')
0013 %           result = eq(c1,c2, 'class/property', 'class/property')
0014 %
0015 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0016 %           result = eq(c1,c2, 'ao/name')
0017 %
0018 % INPUTS:    c1,c2      - input param objects
0019 %           exc_list - exception list
0020 %           List of properties which are not checked.
0021 %
0022 % OUTPUTS:   If the two param objects are considered equal, result == 1,
0023 %           otherwise, result == 0.
0024 %
0025 % VERSION:   $Id: eq.m,v 1.5 2007/12/11 17:27:09 ingo Exp $
0026 %
0027 % HISTORY:   29-08-2007 M Hewitson
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION = '$Id: eq.m,v 1.5 2007/12/11 17:27:09 ingo Exp $';
0033
0034 % Check if this is a call for parameters
0035 if nargin == 2
0036     if isa(c1, 'param') && ischar(c2)
0037         in = char(c2);
0038         if strcmp(in, 'Params')
0039             result = plist;
0040             return

```

```

0041     elseif strcmp(in, 'Version')
0042         result = VERSION;
0043         return
0044     end
0045 end
0046 end
0047
0048 result = 1;
0049
0050 %% Check class
0051 if ~strcmp(class(c1), class(c2))
0052     result = 0;
0053     return
0054 end
0055
0056 fields = fieldnames(c1);
0057
0058 for ii = 1:length(fields)
0059     field = fields{ii};
0060
0061     ck_field = {field, ['param/' field]};
0062
0063     % Is the field equal to value then add the name 'value'
0064     % to the exception list.
0065     if strcmp(field, 'val')
0066         ck_field{end+1} = 'value';
0067         ck_field{end+1} = 'param/value';
0068     end
0069
0070 %% Check tag
0071 if ~(any(ismember(ck_field, varargin)))
0072
0073     if isobject(c1.(field))
0074         if ~eq(c1.(field), c2.(field), varargin{:})
0075             result = 0;
0076             disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0077             return
0078         end
0079     else
0080         if ~isequalwithhequalnans(c1.(field), c2.(field))
0081             result = 0;
0082             disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0083             return
0084         end
0085     end
0086
0087 end
0088 end
0089

```

[Home](#) > [m](#) > [sigproc](#) > [time_domain](#) > ltpda_polydetrend.m

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 1.6 2007/11/08 10:35:04 hewitson 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

```

0001 function varargout = ltpda_polydetrend(varargin)
0002 % LTPDA_POLYDETREND detrends the input analysis object using a polynomial of
0003 % degree N.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_POLYDETREND detrends the input analysis object using a
0007 %               polynomial of degree N.
0008 %
0009 % CALL:        b = ltpda_polydetrend(a1,a2,a3,..., pl)
0010 %
0011 % INPUTS:      aN  - a list of analysis objects
0012 %               pl  - a parameter list
0013 %
0014 % OUTPUTS:     b - array of analysis objects
0015 %
0016 %               If the last input argument is a parameter list (plist) it is
0017 %               used.
0018 %               The following parameters are recognised.
0019 %
0019 % PARAMETERS:  'N' - degree of polynomial [default: 1]
0020 %
0021 % GUI CALL:    The following call returns a parameter list object that contains
0022 %               the default parameter values:
0023 %
0024 %               >> pl = ltpda_xcorr('Params')
0025 %
0026 % VERSION:     $Id: ltpda_polydetrend.m,v 1.6 2007/11/08 10:35:04 hewitson Exp $
0027 %
0028 % HISTORY:     01-03-2007 M Hewitson
0029 %               Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032 %
0033 % Check if this is a call for parameters
0034 if nargin == 1
0035     in = char(varargin{1});
0036     if strcmp(in, 'Params')
0037         varargout{1} = getDefaultPL();
0038     return

```

```

0039     end
0040 end
0041
0042 % capture input variable names
0043 invars = {};
0044 as      = [];
0045 ps      = [];
0046 for j=1:nargin
0047     invars = [invars cellstr(inputname(j))];
0048     if isa(varargin{j}, 'ao')
0049         as = [as varargin{j}];
0050     end
0051     if isa(varargin{j}, 'plist')
0052         ps = [ps varargin{j}];
0053     end
0054 end
0055
0056 ALGONAME = mfilename;
0057 VERSION  = '$Id: ltpda_polydetrend.m,v 1.6 2007/11/08 10:35:04 hewitson Exp $';
0058
0059 % Look at inputs
0060 if nargin < 1
0061     error('### Incorrect number of inputs.')
0062 end
0063
0064 na = length(as);
0065
0066 % Combine plists
0067 if ~isempty(ps)
0068     pl = combine(ps, getDefaultPL());
0069 else
0070     pl = getDefaultPL();
0071 end
0072
0073 % Initialise output
0074 bo = [];
0075
0076 % Unpack parameter list
0077 N = find(pl, 'N');
0078
0079 % Loop over analysis objects
0080 for i=1:na
0081
0082     % get data out
0083     a = as(i);
0084     d = a.data;
0085     dinfo = whos('d');
0086     if ~isa(d, 'tsdata')
0087         error('### I only work with time-series at the moment.');
```

```

0092
0093     y = polydetrend(d.y, N);
0094
0095     % Make output analysis object
0096     nameStr = sprintf('poly_detrend(%s)', d.name);
0097
0098     % create new output data
0099     data = tsdata(y, d.fs);
0100     data = set(data, 'name', nameStr);
0101     data = set(data, 'xunits', d.xunits);
0102     data = set(data, 'yunits', d.yunits);
0103
0104     % create new output history
0105     h = history(ALGONAME, VERSION, pl, a.hist);
0106     h = set(h, 'invars', invars);
0107
0108     % make output analysis object
0109     b = ao(data, h);
0110
0111     % set name
0112     % name for this object
0113     if isempty(invars{i})
0114         n1 = a.name;
0115     else
0116         n1 = invars{i};
0117     end
0118
0119     nameStr = sprintf('poly_detrend(%s)', n1);
0120     b = setnh(b, 'name', nameStr);
0121
0122     % Add to output array
0123     bo = [bo b];
0124
0125 end
0126
0127 varargout{1} = bo;
0128
0129 %-----
0130 % Get default params
0131 function plo = getDefaultPL()
0132
0133 disp('* creating default plist...');
0134 plo = plist();
0135 plo = append(plo, param('N', 1));
0136 disp('* done. ');
0137
0138 % END

```

[Home](#) > [classes](#) > [@param](#) > set.m

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 1.8 2007/11/23 15:34:47 ingo Exp \$

HISTORY: 30-01-07 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

```
0001 function p = set(p, varargin)
0002 % SET set a parameter property.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET set a parameter property.
0007 %
0008 % CALL: p = set(p, 'key',      'name');
0009 %       p = set(p, 'val',      val);
0010 %       p = set(p, 'val',      'val');
0011 %       p = set(p, 'name',     'my name');
0012 %       p = set(p, 'created',  time-object);
0013 %
0014 % VERSION: $Id: set.m,v 1.8 2007/11/23 15:34:47 ingo Exp $
0015 %
0016 % HISTORY: 30-01-07 M Hewitson
0017 %           Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: set.m,v 1.8 2007/11/23 15:34:47 ingo Exp $';
0022
0023 % Check if this is a call for parameters or for the cvs-version number
0024 if nargin == 2
0025     if isa(p, 'param') && ischar(varargin{1})
0026         in = char(varargin{1});
0027         if strcmp(in, 'Params')
0028             p = plist();
0029             return
0030         elseif strcmp(in, 'Version')
0031             p = VERSION;
0032             return
0033         end
0034     end
0035 end
0036
0037
0038 if (nargout ~= 0)
0039
0040     propArgin = varargin;
0041
0042     while length(propArgin) >= 2
0043         prop = varargin{1};
0044         val = varargin{2};
0045         propArgin = propArgin(3:end);
0046         switch prop
0047             case 'name'
0048                 p.name = val;
0049             case 'version'
0050                 p.version = val;
0051             case 'created'
0052                 if isa(val, 'time')
0053                     p.created = val;
0054                 else
0055                     error('### The property ''created'' must be a time-object');

```

```
0056         end
0057     case 'key'
0058         if ischar(val)
0059             p.key = upper(val);
0060         else
0061             error('### The Key of a parameter must be a string. ');
0062         end
0063     case 'val'
0064         p.val = val;
0065     otherwise
0066         error([val, ' is not a property of param class']);
0067     end
0068 end
0069
0070 else
0071     error('### please use: param = set(param, ''key'', ''value'');');
0072 end
0073
```

[Home](#) > [classes](#) > [@cdata](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a cdata from a cdata xml file.

SYNOPSIS

function ps = xmlparse(p, xmlfile)

DESCRIPTION

XMLPARSE creates a cdata from a cdata xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a cdata object into a string.

This function is called by:

- [cdata](#) CDATA constant data object class constructor.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a cdata from a cdata xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```

```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'cdata')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```


ltpda_psd

PURPOSE

LTPDA_PSD a psd estimation based on pwelch from MATLAB.

SYNOPSIS

function varargout = ltpda_psd(x, win, nlap, nfft, fs, scale, inunits)

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 linear detrending is done on each
             windowed data segment.

CALL:       [f, pxx, info] = ltpda_psd(x, win, nlap, 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
            nlap    - 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.

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 1.2 2007/07/16 12:52:21 ingo 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

```

0001 function varargout = ltpda_psd(x, win, nolap, nfft, fs, scale, inunits)
0002 % LTPDA_PSD a psd estimation based on pwelch from MATLAB.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_PSD a psd estimation based on pwelch from MATLAB.
0007 %               The difference is that linear detrending is done on each
0008 %               windowed data segment.
0009 %
0010 % CALL:         [f, pxx, info] = ltpda_psd(x, win, nolap, nfft, fs,
scale, inunits)
0011 %
0012 % INPUTS:       x      - data vector
0013 %               win    - a window function
0014 %               Specify a vector of window samples or a length. If
a length
0015 %               is specified, a Hamming window is used.
0016 %               nfft   - number of samples per fft
0017 %               nolap  - number of samples to overlap each segment by
0018 %               fs     - sample rate of input time-series
0019 %               scale  - specify one of 'ASD', 'AS', 'PSD', 'PS'
0020 %               inunits - Amplitude units of input time-series data.
0021 %
0022 % OUTPUTS:      f      - a frequency vector
0023 %               Sxx    - spectral estimate in units specified by 'scale'
0024 %               info   - a structure of information
0025 %
0026 % NOTE:         segment_info() is copied directly from MATLAB's welshparse.m.
0027 %
0028 % VERSION:      $Id: ltpda_psd.m,v 1.2 2007/07/16 12:52:21 ingo Exp $
0029 %
0030 % HISTORY:      25-09-2006 M Hewitson
0031 %               Creation
0032 %
0033 % The following call returns a parameter list object that contains the
0034 % default parameter values:
0035 %
0036 % >> pl = ltpda_psd('Params')
0037 %
0038 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0039
0040 % get segment information
0041 Nx = length(x);
0042 [L,noverlap,win,msg] = segment_info(Nx,win,nolap);
0043 % Compute the number of segments
0044 k = (Nx-noverlap)./(L-noverlap);

```

```

0045
0046 % Uncomment the following line to produce a warning each time the data
0047 % segmentation does not produce an integer number of segments.
0048 if fix(k) ~= k
0049     warning('The number of segments is not an integer, truncating data.');
```

```

0050 end
0051 k = fix(k);
0052
0053 % fix vectors
0054 x    = x(:);
0055 win = win(:);
0056
0057 % Initialise the spectral estimates
0058 Sxx = zeros(nfft,1);
0059
0060 % loop over segments
0061 LminusOverlap = L-noverlap;
0062 xStart = 1:LminusOverlap:k*LminusOverlap;
0063 xEnd    = xStart+L-1;
0064
0065 for i = 1:k,
0066     %disp(sprintf('* computing segment %d', i));
0067     % data segment, detrend and window
0068     dseg = detrend(x(xStart(i):xEnd(i))) .* win;
0069     %dseg = (x(xStart(i):xEnd(i))) .* win;
0070
0071     % compute fft
0072     xx    = fft(dseg,nfft);
0073     Sxx = ((Sxx .* (i-1)) + xx.*conj(xx)) ./ i;
0074 end
0075 % take only one-sided
0076 Sxx = Sxx(1:nfft/2+1);
0077
0078 % scale the data as required
0079 [Sxx, info] = scale_xx(Sxx, win, fs, scale, inunits);
0080
0081 % grow frequency vector
0082 f = linspace(0, fs/2, length(Sxx));
0083
0084 % complete info
0085 info.nsecs = Nx/fs;
0086 info.navs  = k;
0087 info.nolap = noverlap/fs;
0088
0089 if nargout == 3
0090
0091     varargout{1} = f;
0092     varargout{2} = Sxx;
0093     varargout{3} = info;
0094
0095
0096 else
0097     disp('usage: [f, pxx, info] = ltpda_psd(x, win, nfft, nolap, fs)');
0098     error('# incorrect usage');
0099 end
0100
0101
0102
%-----

```

```

0103 function [yy, info] = scale_xx(xx, win, fs, norm, inunits)
0104
0105 nfft = length(win);
0106 S1 = sum(win);
0107 S2 = sum(win.^2);
0108 enbw = fs * S2 / (S1*S1);
0109
0110 switch norm
0111     case 'ASD'
0112         yy = sqrt(xx) * sqrt(2 / (fs*S2));
0113         info.units = [sprintf('%s', inunits) '\surdHz'];
0114     case 'PSD'
0115         yy = xx * 2 / (fs*S2);
0116         info.units = sprintf('%s^2/Hz', inunits);
0117     case 'AS'
0118         yy = sqrt(xx) * sqrt(2) / S1;
0119         info.units = sprintf('%s', inunits);
0120     case 'PS'
0121         yy = xx * 2 / (S1^2);
0122         info.units = sprintf('%s^2', inunits);
0123     case 'none'
0124         yy = xx;
0125         info.units = '';
0126     otherwise
0127         error('Unknown normalisation');
0128 end
0129
0130 info.nfft = nfft;
0131 info.enbw = enbw;
0132 info.norm = norm;
0133
0134
%-----
0135 function [L,noverlap,win,msg] = segment_info(M,win,noverlap)
0136 %SEGMENT_INFO Determine the information necessary to segment the input data.
0137 %
0138 % Inputs:
0139 %     M - An integer containing the length of the data to be segmented
0140 %     WIN - A scalar or vector containing the length of the window or
the window respectively
0141 %         (Note that the length of the window determines the length
of the segments)
0142 %     NOVERLAP - An integer containing the number of samples to overlap
(may be empty)
0143 %
0144 % Outputs:
0145 %     L - An integer containing the length of the segments
0146 %     NOVERLAP - An integer containing the number of samples to overlap
0147 %     WIN - A vector containing the window to be applied to each section
0148 %     MSG - A string containing possible error messages
0149 %
0150 %
0151 % The key to this function is the following equation:
0152 %
0153 %     
$$K = (M - NOVERLAP) / (L - NOVERLAP)$$

0154 %
0155 % where
0156 %
0157 %     K - Number of segments

```

```

0158 %      M      - Length of the input data X
0159 %      NOVERLAP - Desired overlap
0160 %      L      - Length of the segments
0161 %
0162 % The segmentation of X is based on the fact that we always know M and two
of the set
0163 % {K,NOVERLAP,L}, hence determining the unknown quantity is trivial from
the above
0164 % formula.
0165
0166 % Initialize outputs
0167 L = [];
0168 msg = '';
0169
0170 % Check that noverlap is a scalar
0171 if any(size(noverlap) > 1),
0172     msg.identifier = generatemsgid('invalidNoverlap');
0173     msg.message = 'You must specify an integer number of samples to overlap.';
0174     return;
0175 end
0176
0177 if isempty(win),
0178     % Use 8 sections, determine their length
0179     if isempty(noverlap),
0180         % Use 50% overlap
0181         L = fix(M./4.5);
0182         noverlap = fix(0.5.*L);
0183     else
0184         L = fix((M+7.*noverlap)./8);
0185     end
0186     % Use a default window
0187     win = hamming(L);
0188 else
0189     % Determine the window and its length (equal to the length of the segments)
0190     if ~any(size(win) <= 1) | ischar(win),
0191         msg.identifier = generatemsgid('invalidWindow');
0192         msg.message = 'The WINDOW argument must be a vector or a scalar.';
0193         return
0194     elseif length(win) > 1,
0195         % WIN is a vector
0196         L = length(win);
0197     elseif length(win) == 1,
0198         L = win;
0199         win = hamming(win);
0200     end
0201     if isempty(noverlap),
0202         % Use 50% overlap
0203         noverlap = fix(0.5.*L);
0204     end
0205 end
0206
0207 % Do some argument validation
0208 if L > M,
0209     msg.identifier = generatemsgid('invalidSegmentLength');
0210     msg.message = 'The length of the segments cannot be greater than the
length of the input signal.';
0211     return;
0212 end
0213
0214 if noverlap >= L,

```

```
0215     msg.identifier = generatemsgid('invalidNoverlap');
0216     msg.message = 'The number of samples to overlap must be less than
the length of the segments.';
0217     return;
0218 end
0219
0220
%- -----
0221
0222 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@fsdata](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a fsdata from a fsdata xml file.

SYNOPSIS

function ps = xmlparse(p, xmlfile)

DESCRIPTION

XMLPARSE creates a fsdata from a fsdata xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a param object into a string.

This function is called by:

- [fsdata](#) FSDATA frequency-series object class constructor.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a fsdata from a fsdata xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```

```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'fsdata')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```


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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(c1,c2, 'ao/name')

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 1.3 2008/01/02 17:58:51 ingo Exp \$

HISTORY: 29-08-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- **char** CHAR convert a pole object into a string.
- **eq** EQ overloads the == operator for pole objects.

This function is called by:

- **eq** EQ overloads the == operator for pole objects.
- **ne** NE overloads the ~= operator for pole objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002 % EQ overloads the == operator for pole objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EQ overloads the == operator for pole objects.
0007 %
0008 %           All fields are checked.
0009 %
0010 % CALL:      result = eq(c1,c2)
0011 %           result = eq(c1,c2, exc_list)
0012 %           result = eq(c1,c2, 'property1', 'property2')
0013 %           result = eq(c1,c2, 'class/property', 'class/property')
0014 %
0015 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0016 %           result = eq(c1,c2, 'ao/name')
0017 %
0018 % INPUTS:    c1,c2      - input pole objects
0019 %           exc_list - exception list
0020 %           List of properties which are not checked.
0021 %
0022 % OUTPUTS:   If the two pole objects are considered equal, result == 1,
0023 %           otherwise, result == 0.
0024 %
0025 % VERSION:   $Id: eq.m,v 1.3 2008/01/02 17:58:51 ingo Exp $
0026 %
0027 % HISTORY:   29-08-2007 M Hewitson
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION = '$Id: eq.m,v 1.3 2008/01/02 17:58:51 ingo Exp $';
0033
0034 % Check if this is a call for parameters
0035 if nargin == 2
0036     if isa(c1, 'pole') && ischar(c2)
0037         in = char(c2);
0038         if strcmp(in, 'Params')
0039             result = plist;
0040             return

```

```

0041     elseif strcmp(in, 'Version')
0042         result = VERSION;
0043         return
0044     end
0045 end
0046 end
0047
0048 result = 1;
0049
0050 %% Check class
0051 if ~strcmp(class(c1), class(c2))
0052     result = 0;
0053     return
0054 end
0055
0056 fields = fieldnames(c1);
0057
0058 for ii = 1:length(fields)
0059     field = fields{ii};
0060
0061     ck_field = {field, ['pole/' field]};
0062
0063     %% Check tag
0064     if ~(any(ismember(ck_field, varargin)))
0065
0066         if isobject(c1.(field))
0067
0068             if length(c1.(field)) ~= length(c2.(field))
0069                 result = 0;
0070                 return
0071             end
0072
0073             for jj = 1:length(c1.(field))
0074                 if ~eq(c1.(field)(jj), c2.(field)(jj), varargin{:})
0075                     result = 0;
0076                     disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0077                     return
0078                 end
0079             end
0080         else
0081             if ~isequalwithhequalnans(c1.(field), c2.(field))
0082                 result = 0;
0083                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0084                 return
0085             end
0086         end
0087     end
0088 end
0089 end
0090

```


[Home](#) > [m](#) > [sigproc](#) > [frequency_domain](#) > ltpda_pwelch.m

ltpda_pwelch

PURPOSE

LTPDA_PWELCH makes power spectral density estimates of the time-series objects

SYNOPSIS

function varargout = ltpda_pwelch(varargin)

DESCRIPTION

LTPDA_PWELCH makes power spectral density estimates of the time-series objects

%%%

DESCRIPTION: LTPDA_PWELCH makes power spectral density estimates of the time-series objects in the input analysis objects.

CALL: b = ltpda_pwelch(a1,a2,a3,...,pl)

INPUTS: b - output analysis objects
 aN - input analysis objects
 pl - input parameter list

Makes PSD estimates using ltpda_psd() of each input A0.

If the last input argument is a parameter list (plist) it is used.
 The following parameters are recognised.

PARAMETERS: 'Win' - a specwin window object [default: Kaiser -200dB psll]
 '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_pwelch.m,v 1.15 2007/11/08 10:35:04 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')
```

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [ltpda_psd](#) LTPDA_PSD a psd estimation based on pwelch from MATLAB.

This function is called by:

SUBFUNCTIONS

- [function plo = getDefaultPL\(\)](#)

SOURCE CODE

```

0001 function varargout = ltpda_pwelch(varargin)
0002 % LTPDA_PWELCH makes power spectral density estimates of the time-series objects
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_PWELCH makes power spectral density estimates of the
0007 %               time-series objects in the input analysis objects.
0008 %
0009 % CALL:        b = ltpda_pwelch(a1,a2,a3,...,pl)
0010 %
0011 % INPUTS:      b      - output analysis objects
0012 %              aN     - input analysis objects
0013 %              pl     - input parameter list
0014 %
0015 %              Makes PSD estimates using ltpda_psd() of each input A0.
0016 %
0017 %              If the last input argument is a parameter list (plist) it is used.
0018 %              The following parameters are recognised.
0019 %
0020 % PARAMETERS: 'Win'    - a specwin window object [default: Kaiser -200dB psll]
0021 %              'Nolap' - segment overlap (number of samples) [default: taken
from window function]
0022 %              'Nfft'  - number of samples in each fft [default: fs of input
data]
0023 %              'Debug' - debug level for terminal output (0-1)
0024 %
0025 % VERSION:     $Id: ltpda_pwelch.m,v 1.15 2007/11/08 10:35:04 hewitson Exp $
0026 %
0027 % HISTORY:     07-02-2007 M Hewitson
0028 %              Creation
0029 %
0030 % The following call returns a parameter list object that contains the
0031 % default parameter values:
0032 %
0033 % >> pl = ltpda_pwelch('Params')

```

```

0034 %
0035 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0036
0037 ALGONAME = mfilename;
0038 VERSION  = '$Id: ltpda_pwelch.m,v 1.15 2007/11/08 10:35:04 hewitson Exp $';
0039
0040 % Initialise output
0041 bs = [];
0042
0043 % Check if this is a call for parameters
0044 if nargin == 1
0045     in = char(varargin{1});
0046     if strcmp(in, 'Params')
0047         varargout{1} = getDefaultPL();
0048         return
0049     end
0050 end
0051
0052 % Read inputs
0053 invars = {};
0054 as     = [];
0055 pls    = [];
0056 for j=1:nargin
0057     if isa(varargin{j}, 'ao')
0058         as = [as varargin{j}];
0059         invars = [invars cellstr(inputname(j))];
0060     end
0061     if isa(varargin{j}, 'plist')
0062         pls = [pls varargin{j}];
0063     end
0064 end
0065 if isempty(pls)
0066     pls = plist();
0067 end
0068 na = length(as);
0069
0070 % Loop over input AOs
0071 for j=1:na
0072
0073     % Unpack AO
0074     a = as(j);
0075     d = a.data;
0076
0077     if isa(a, 'ao')
0078         % check input data
0079         if isa(d, 'tsdata')
0080
0081             % combine plists
0082             plo = combine(pls, getDefaultPL());
0083
0084             % check all defaults are sensible
0085             Nfft = find(plo, 'Nfft'); % Number of points in each fft
0086             setWindow = 0;

```

```

0087     if Nfft <= 0
0088         Nfft = length(d.y);
0089         disp(sprintf('! Using default Nfft of %g', Nfft))
0090         setWindow = 1;
0091     end
0092     plo = pset(plo, 'Nfft', Nfft);
0093
0094     Win = find(plo, 'Win');    % Window object
0095     if setWindow || length(Win.win)~=Nfft
0096         switch Win.name
0097             case {'Kaiser', 'Flattop'}
0098                 Win = specwin(Win.name, Nfft, Win.psl1);
0099             otherwise
0100                 Win = specwin(Win.name, Nfft);
0101         end
0102         disp(sprintf('! Reset window to %s(%d)', strrep(Win.name, '_', '\_'),
length(Win.win)))
0103     end
0104     plo = pset(plo, 'Win', Win);
0105
0106     Noverlap = find(plo, 'Noverlap');    % Amount to overlap each fft
0107     if Noverlap <= 0
0108         Noverlap = round(Win.rov*Nfft/100);
0109         disp(sprintf('! Using default overlap of %d samples', Noverlap))
0110     end
0111     plo = pset(plo, 'Noverlap', Noverlap);
0112
0113     % check parameters
0114     if length(Win.win) ~= Nfft
0115         error('### Window length should be same as Nfft.');
```

```

0116     end
0117
0118     % Compute PSD using pwelch
0119     [f, pxx, info] = ltpda_psd(d.y, Win.win, Noverlap, Nfft, d.fs, 'ASD', d.
yunits);
0120
0121     % create new output fsdata
0122     fs = fsdata(f, pxx, d.fs);
0123     fs = set(fs, 'name', sprintf('PSD(%s)', d.name));
0124     fs = set(fs, 'yunits', info.units);
0125     fs = set(fs, 'enbw', Win.nenbw*d.fs/Nfft);
0126     fs = set(fs, 'navs', info.navs);
0127
0128     % create new output history
0129     h = history(ALGONAME, VERSION, plo, a.hist);
0130     h = set(h, 'invars', invars);
0131
0132     % make output analysis object
0133     b = ao(fs, h);
0134
0135     % set name
0136     if na~=length(invars) || isempty(invars{j})
0137         nj = a.name;
```



```

0138     else
0139         nj = invars{j};
0140     end
0141     b = setnh(b, 'name', sprintf('PSD(%s)', nj));
0142
0143     % add to outputs
0144     bs = [bs b];
0145
0146     else
0147         warning('### Ignoring input AO number %d (%s); it is not a time-series.',
j, a.name)
0148     end
0149 end
0150 end % loop over analysis objects
0151
0152 % set output
0153 varargout{1} = bs;
0154
0155
0156 %-----
0157 % Get default params
0158 function plo = getDefaultPL()
0159
0160 disp('* creating default plist...');
0161 plo = plist();
0162 plo = append(plo, param('Nfft', -1));
0163 plo = append(plo, param('Win', specwin('Kaiser', 1, 200)));
0164 plo = append(plo, param('Nolap', -1));
0165 disp('* done. ');
0166
0167
0168

```

[Home](#) > [classes](#) > [@pole](#) > set.m

set

PURPOSE

SET set a pole property.

SYNOPSIS

function p = set(p, varargin)

DESCRIPTION

SET set a pole property.

%%%

DESCRIPTION: SET a history property.

CALL: p = set(p, 'name', 'name');
 p = set(p, 'f', f);
 p = set(p, 'q', q);
 p = set(p, 'ri', ri);
 p = set(p, 'created', time-object);
 p = set(p, 'version', 'plist/set.m,v 1.7...');

VERSION: \$Id: set.m,v 1.7 2007/10/16 17:37:21 ingo Exp \$

HISTORY: 30-01-07 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

```
0001 function p = set(p, varargin)
0002 % SET set a pole property.
```

```

0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET a history property.
0007 %
0008 % CALL:          p = set(p, 'name',      'name');
0009 %                p = set(p, 'f',        f);
0010 %                p = set(p, 'q',        q);
0011 %                p = set(p, 'ri',       ri);
0012 %                p = set(p, 'created',  time-object);
0013 %                p = set(p, 'version', 'plist/set.m,v 1.7...');
0014 %
0015 % VERSION:        $Id: set.m,v 1.7 2007/10/16 17:37:21 ingo Exp $
0016 %
0017 % HISTORY:        30-01-07 M Hewitson
0018 %                  Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: set.m,v 1.7 2007/10/16 17:37:21 ingo Exp $';
0023
0024 % Check if this is a call for parameters or for the cvs-version number
0025 if nargin == 2
0026     if isa(p, 'pole') && ischar(varargin{1})
0027         in = char(varargin{1});
0028         if strcmp(in, 'Params')
0029             p = plist();
0030             return
0031         elseif strcmp(in, 'Version')
0032             p = VERSION;
0033             return
0034         end
0035     end
0036 end
0037
0038 if (nargout ~= 0)
0039
0040     propArgin = varargin;
0041     while length(propArgin) >= 2
0042         prop = varargin{1};
0043         val = varargin{2};
0044         propArgin = propArgin(3:end);
0045         switch prop
0046             case 'name'
0047                 p.name = val;
0048             case 'version'
0049                 p.version = val;
0050             case 'created'
0051                 p.created = val;
0052             case 'f'
0053                 p.f = val;
0054                 p.ri = pfq2ri(p.f, p.q);

```

```

0055     case 'q'
0056         p.q = val;
0057         p.ri = pfq2ri(p.f, p.q);
0058     case 'ri'
0059         if length(val) == 1
0060             p.ri = [val conj(val)];
0061         else
0062             p.ri = val;
0063         end
0064         [p.f, p.q] = pri2fq(p.ri);
0065     otherwise
0066         error([val, ' is not a property of pole class']);
0067     end
0068 end
0069
0070 else
0071     if ischar(varargin{2})
0072         error('### please use: %s = set(%s, '%s'', '%s'');', ...
0073             inputname(1), ...
0074             inputname(1), ...
0075             varargin{1}, ...
0076             varargin{2});
0077     elseif isnumeric(varargin{2})
0078         error('### please use: %s = set(%s, '%s'', %d);', ...
0079             inputname(1), ...
0080             inputname(1), ...
0081             varargin{1}, ...
0082             varargin{2});
0083     else
0084         error('### please use: %s = set(%s, key, 'value'');', ...
0085             inputname(1), ...
0086             inputname(1));
0087     end
0088 end

```

[Home](#) > [classes](#) > [@history](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a history from a history xml file.

SYNOPSIS

function ps = xmlparse(p, xmlfile)

DESCRIPTION

XMLPARSE creates a history from a history xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a param object into a string.

This function is called by:

- [history](#) HISTORY History object class constructor.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a history from a history xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```

```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'history')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```

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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')

result = eq(c1,c2, 'ao/name')

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 1.4 2008/01/07 10:21:53 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:

- **eq** EQ overloads the == operator for provenance objects.
- **ne** NE overloads the ~= operator for provenance objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002 % EQ overloads the == operator for provenance objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EQ overloads the == operator for provenance objects.
0007 %
0008 %           All fields are checked.
0009 %
0010 % CALL:      result = eq(c1,c2)
0011 %           result = eq(c1,c2,  exc_list)
0012 %           result = eq(c1,c2, 'property1', 'property2')
0013 %           result = eq(c1,c2, 'class/property', 'class/property')
0014 %
0015 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0016 %           result = eq(c1,c2, 'ao/name')
0017 %
0018 % INPUTS:    c1,c2      - input provenance objects
0019 %           exc_list - exception list
0020 %           List of properties which are not checked.
0021 %
0022 % OUTPUTS:   If the two provenance objects are considered equal, result == 1,
0023 %           otherwise, result == 0.
0024 %
0025 % VERSION:   $Id: eq.m,v 1.4 2008/01/07 10:21:53 ingo Exp $
0026 %
0027 % HISTORY:   29-08-2007 M Hewitson
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION = '$Id: eq.m,v 1.4 2008/01/07 10:21:53 ingo Exp $';
0033
0034 % Check if this is a call for parameters
0035 if nargin == 2
0036     if isa(c1, 'provenance') && ischar(c2)
0037         in = char(c2);
0038         if strcmp(in, 'Params')
0039             result = plist;
0040             return
0041         elseif strcmp(in, 'Version')

```



```

0042     result = VERSION;
0043     return
0044 end
0045 end
0046 end
0047
0048 result = 1;
0049
0050 %% Check class
0051 if ~strcmp(class(c1), class(c2))
0052     result = 0;
0053     return
0054 end
0055
0056 fields = fieldnames(c1);
0057
0058 for ii = 1:length(fields)
0059     field = fields{ii};
0060
0061     ck_field = {field, ['provenance/' field]};
0062
0063     %% Check tag
0064     if ~(any(ismember(ck_field, varargin)))
0065
0066         if isobject(c1.(field))
0067             if ~eq(c1.(field), c2.(field), varargin{:})
0068                 result = 0;
0069                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0070                 return
0071             end
0072         else
0073             if ~isequalwithhequalnans(c1.(field), c2.(field))
0074                 result = 0;
0075                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0076                 return
0077             end
0078         end
0079     end
0080 end
0081 end
0082
0083
0084 % function result = eq(p1,p2, varargin)
0085 %
0086 % % EQ overloads the == operator for provenance objects.
0087 % %
0088 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0089 % %
0090 % % DESCRIPTION: EQ overloads the == operator for provenance objects.
0091 % %
0092 % %           All fields are checked.
0093 % %

```

```

0094 % % CALL:      result = eq(w1,w2)
0095 % %
0096 % % INPUTS:    w1,w2 - input specwin objects
0097 % %
0098 % % OUTPUTS:    If the two specwin objects are considered equal, result == 1,
0099 % %               otherwise, result == 0.
0100 % %
0101 % % VERSION:    $Id: eq.m,v 1.4 2008/01/07 10:21:53 ingo Exp $
0102 % %
0103 % % HISTORY:    29-08-2007 M Hewitson
0104 % %               Creation
0105 % %
0106 % %%%%%%%%%%%%%%%
%
0107 %
0108 % result = 1;
0109 %
0110 % % s1 = struct(p1);
0111 % % s2 = struct(p2);
0112 %
0113 % fn1 = fieldnames(p1);
0114 %
0115 % for j=1:length(fn1)
0116 %     field = fn1{j};
0117 %     if ~ismember(field, varargin)
0118 %         try
0119 %             if p1.(field) ~= p2.(field)
0120 %                 result = 0;
0121 %                 return;
0122 %             end
0123 %         catch
0124 %             result = 0;
0125 %             return
0126 %         end
0127 %     end
0128 % end
0129 %
0130 % % result = isequal(s1, struct(p2));
0131 %

```

[Home](#) > [m](#) > [helper](#) > ltpda_rand.m

ltpda_rand

PURPOSE [^](#)

LTPDA_RANDOM return a random number between r1 and r2

SYNOPSIS [^](#)

function val = ltpda_rand(r1, r2)

DESCRIPTION [^](#)

LTPDA_RANDOM return a random number between r1 and r2

%%%

DESCRIPTION: LTPDA_RANDOM 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 [^](#)

```
0001 function val = ltpda_rand(r1, r2)
0002 % LTPDA_RANDOM return a random number between r1 and r2
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
0005 %
0006 % DESCRIPTION: LTPDA_RAND return a random number between r1 and r2
0007 %
0008 % CALL:          val = ltpda_rand(r1, r2)
0009 %
0010 % INPUTS:         r1 - lower limit of the random values
0011 %                 r2 - upper limit of the random values
0012 %
0013 % OUTPUTS:        val - random numbers
0014 %
0015 % VERSION: $Id: ltpda_rand.m,v 1.2 2007/07/13 12:17:39 ingo Exp $
0016 %
0017 % HISTORY: 26-01-2007 M Hewitson
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 int = r2-r1;
0023 rand('state',sum(100*clock));
0024 pause(0.001);
0025 val = r1 + int*rand;
0026
0027
0028 % END
0029
```

[Home](#) > [classes](#) > [@provenance](#) > set.m

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 1.4 2008/01/07 13:48:02 ingo Exp \$

HISTORY: 30-01-2007 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

```

0001 function p = set(p, varargin)
0002 % SET set an provenance property.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET set an provenance property.
0007 %
0008 % CALL:          prov = set(prov, 'creator',          'unknown');
0009 %               prov = set(prov, 'created',           time());
0010 %               prov = set(prov, 'ip',                '127.0.0.2');
0011 %               prov = set(prov, 'hostname',          'hws165');
0012 %               prov = set(prov, 'os',                'GLNX86');
0013 %               prov = set(prov, 'matlab_version',    '7.4 (R2007a)');
0014 %               prov = set(prov, 'matlab_version',    '6.7 (R2007a)');
0015 %               prov = set(prov, 'matlab_version',    '0.4 (R2007a)');
0016 %
0017 % VERSION:       $Id: set.m,v 1.4 2008/01/07 13:48:02 ingo Exp $
0018 %
0019 % HISTORY:       30-01-2007 M Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: set.m,v 1.4 2008/01/07 13:48:02 ingo Exp $';
0025
0026 % Check if this is a call for parameters or for the cvs-version number
0027 if nargin == 2
0028     if isa(p, 'provenance') && ischar(varargin{1})
0029         in = char(varargin{1});
0030         if strcmp(in, 'Params')
0031             p = plist();
0032             return
0033         elseif strcmp(in, 'Version')
0034             p = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 propArgin = varargin;
0041
0042 while length(propArgin) >= 2
0043     prop = propArgin{1};
0044     val = propArgin{2};
0045     propArgin = propArgin(3:end);
0046     switch prop
0047     case 'creator'
0048         p.creator = val;
0049     case 'created'
0050         p.created = val;
0051     case 'ip'
0052         p.ip = val;

```

```
0053     case 'hostname'
0054         p.hostname = val;
0055     case 'os'
0056         p.os = val;
0057     case 'matlab_version'
0058         p.matlab_version = val;
0059     case 'sigproc_version'
0060         p.sigproc_version = val;
0061     case 'ltpda_version'
0062         p.ltpda_version = val;
0063     otherwise
0064         error([val, ' is not a property of provenance class']);
0065 end
0066 end
```

[Home](#) > [classes](#) > [@mfir](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a mfir filter from a mfir xml file.

SYNOPSIS

function ps = xmlparse(p, xmlfile)

DESCRIPTION

XMLPARSE creates a mfir filter from a mfir xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a mfir object into a string.

This function is called by:

- [mfir](#) MFIR FIR filter object class constructor.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a mfir filter from a mfir xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```



```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'mfir')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```

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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(c1,c2, 'ao/name')

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 1.5 2008/01/07 18:18:18 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:

- **eq** EQ overloads the == operator for pzmodel objects.
- **ne** NE overloads the ~= operator for pzmodel objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002 % EQ overloads the == operator for pzmodel objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EQ overloads the == operator for pzmodel objects.
0007 %
0008 %           All fields are checked.
0009 %
0010 % CALL:      result = eq(c1,c2)
0011 %           result = eq(c1,c2, exc_list)
0012 %           result = eq(c1,c2, 'property1', 'property2')
0013 %           result = eq(c1,c2, 'class/property', 'class/property')
0014 %
0015 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0016 %           result = eq(c1,c2, 'ao/name')
0017 %
0018 % INPUTS:    c1,c2      - input pzmodel objects
0019 %           exc_list - exception list
0020 %           List of properties which are not checked.
0021 %
0022 % OUTPUTS:   If the two pzmodel objects are considered equal, result == 1,
0023 %           otherwise, result == 0.
0024 %
0025 % VERSION:   $Id: eq.m,v 1.5 2008/01/07 18:18:18 ingo Exp $
0026 %
0027 % HISTORY:   29-08-2007 M Hewitson
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION    = '$Id: eq.m,v 1.5 2008/01/07 18:18:18 ingo Exp $';
0033
0034 % Check if this is a call for parameters
0035 if nargin == 2
0036     if isa(c1, 'pzmodel') && ischar(c2)
0037         in = char(c2);
0038         if strcmp(in, 'Params')
0039             result = plist;
0040             return

```

```

0041     elseif strcmp(in, 'Version')
0042         result = VERSION;
0043         return
0044     end
0045 end
0046 end
0047
0048 result = 1;
0049
0050 %% Check class
0051 if ~strcmp(class(c1), class(c2))
0052     result = 0;
0053     return
0054 end
0055
0056 fields = fieldnames(c1);
0057
0058 for ii = 1:length(fields)
0059     field = fields{ii};
0060
0061     ck_field = {field, ['pzmodel/' field]};
0062
0063     %% Check tag
0064     if ~(any(ismember(ck_field, varargin)))
0065
0066         if isobject(c1.(field))
0067
0068             if length(c1.(field)) ~= length(c2.(field))
0069                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0070                 result = 0;
0071                 return
0072             end
0073
0074             for jj = 1:length(c1.(field))
0075                 if ~eq(c1.(field)(jj), c2.(field)(jj), varargin{:})
0076                     result = 0;
0077                     disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0078                     return
0079                 end
0080             end
0081         else
0082             if ~isequalwithhequalnans(c1.(field), c2.(field))
0083                 result = 0;
0084                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0085                 return
0086             end
0087         end
0088     end
0089 end
0090 end
0091

```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [timetools](#) > ltpda_reformat_date.m

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

```
0001 function s = ltpda_reformat_date (s)
```

```

0002 % LTPDA_REFORMAT_DATE reformats the input date
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_REFORMAT_DATE reformats the input date of the (preferred)
0007 %                form 'yyyy-mm-dd HH:MM:SS' to the format 'mm-dd-yyyy HH:MM:SS'
0008 %                which works with datenum.
0009 %
0010 % CALL:         s  = ltpda_reformat_date(si)
0011 %
0012 % INPUTS:       si - date string in the format 'yyyy-mm-dd HH:MM:SS'
0013 %
0014 % OUTPUTS:      s   - date string in the format 'mm-dd-yyyy HH:MM:SS'
0015 %
0016 % EXAMPLE:      dnum = datenum(ltpda_reformat_date('2007-05-04 22:34:22'));
0017 %
0018 % VERSION: $Id: ltpda_reformat_date.m,v 1.1 2007/07/25 09:21:35 ingo Exp $
0019 %
0020 % HISTORY: 03-06-2007 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 % reformt string to stupid matlab format MM-DD-YYY
0026 s=strcat(s(6:10),'-',s(1:4),s(11:length(s)));
0027
0028

```

[Home](#) > [classes](#) > [@pzmodel](#) > set.m

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: pzmmodel = set(pzmmodel, 'name', str);
 pzmmodel = set(pzmmodel, 'gain', g);
 pzmmodel = set(pzmmodel, 'poles', p);
 pzmmodel = set(pzmmodel, 'zeros', z);
 pzmmodel = set(pzmmodel, 'created', time-object);
 pzmmodel = set(pzmmodel, 'version', cvs version string);

VERSION: \$Id: set.m,v 1.4 2007/10/15 17:16:56 ingo Exp \$

HISTORY: 30-01-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a pzmmodel object into a string.

This function is called by:

- [resp](#) RESP returns the complex response of a pzmmodel as an Analysis Object.
- [tomfir](#) TOMFIR approximates a pole/zero model with an FIR filter.

SOURCE CODE


```

0001 function pzm = set(pzm, varargin)
0002 % SET sets a pole/zero model property.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET sets a pole/zero model property.
0007 %
0008 % CALL:          pzmodel = set(pzmodel, 'name',    str);
0009 %               pzmodel = set(pzmodel, 'gain',     g);
0010 %               pzmodel = set(pzmodel, 'poles',    p);
0011 %               pzmodel = set(pzmodel, 'zeros',    z);
0012 %               pzmodel = set(pzmodel, 'created',  time-object);
0013 %               pzmodel = set(pzmodel, 'version',  cvs version string);
0014 %
0015 % VERSION:       $Id: set.m,v 1.4 2007/10/15 17:16:56 ingo Exp $
0016 %
0017 % HISTORY:       30-01-2007 M Hewitson
0018 %                 Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: set.m,v 1.4 2007/10/15 17:16:56 ingo Exp $';
0023
0024 % Check if this is a call for parameters or for the cvs-version number
0025 if nargin == 2
0026     if isa(pzm, 'pzmodel') && ischar(varargin{1})
0027         in = char(varargin{1});
0028         if strcmp(in, 'Params')
0029             pzm = plist();
0030             return
0031         elseif strcmp(in, 'Version')
0032             pzm = VERSION;
0033             return
0034         end
0035     end
0036 end
0037
0038 propArgin = varargin;
0039 fields    = fieldnames(pzm);
0040
0041 while length(propArgin) >= 2
0042     prop = varargin{1};
0043     val  = varargin{2};
0044     propArgin = propArgin(3:end);
0045
0046     if ~ismember(prop, fields)
0047         error(['### ', prop, ' is not a valid pole/zero model property.']);
0048     else
0049         pzm.(prop) = val;
0050     end
0051

```

0052 [end](#)

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@miir](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a miir filter from a miir xml file.

SYNOPSIS

```
function ps = xmlparse(p, xmlfile)
```

DESCRIPTION

XMLPARSE creates a miir filter from a miir xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a miir object into a string.

This function is called by:

- [miir](#) MIIR IIR filter object class constructor.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a miir filter from a miir xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```

```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'miir')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```

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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(c1,c2, 'ao/name')

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 1.3 2008/01/08 16:40:29 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:

- **eq** EQ overloads the == operator for specfin objects.
- **ne** NE overloads the ~= operator for specwin objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002 % EQ overloads the == operator for specfin objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EQ overloads the == operator for specfin objects.
0007 %
0008 %           All fields are checked.
0009 %
0010 % CALL:      result = eq(c1,c2)
0011 %            result = eq(c1,c2, exc_list)
0012 %            result = eq(c1,c2, 'property1', 'property2')
0013 %            result = eq(c1,c2, 'class/property', 'class/property')
0014 %
0015 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0016 %            result = eq(c1,c2, 'ao/name')
0017 %
0018 % INPUTS:    c1,c2      - input specfin objects
0019 %            exc_list - exception list
0020 %                  List of properties which are not checked.
0021 %
0022 % OUTPUTS:   If the two specfin objects are considered equal, result == 1,
0023 %            otherwise, result == 0.
0024 %
0025 % VERSION:   $Id: eq.m,v 1.3 2008/01/08 16:40:29 ingo Exp $
0026 %
0027 % HISTORY:   29-08-2007 M Hewitson
0028 %            Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION    = '$Id: eq.m,v 1.3 2008/01/08 16:40:29 ingo Exp $';
0033
0034 % Check if this is a call for parameters
0035 if nargin == 2
0036     if isa(c1, 'specwin') && ischar(c2)
0037         in = char(c2);
0038         if strcmp(in, 'Params')
0039             result = plist;
0040             return

```

```
0041     elseif strcmp(in, 'Version')
0042         result = VERSION;
0043         return
0044     end
0045 end
0046 end
0047
0048 result = 1;
0049
0050 %% Check class
0051 if ~strcmp(class(c1), class(c2))
0052     result = 0;
0053     return
0054 end
0055
0056 fields = fieldnames(c1);
0057
0058 for ii = 1:length(fields)
0059     field = fields{ii};
0060
0061     ck_field = {field, ['specwin/' field]};
0062
0063     %% Check tag
0064     if ~(any(ismember(ck_field, varargin)))
0065
0066         if isobject(c1.(field))
0067             if ~eq(c1.(field), c2.(field), varargin{:})
0068                 result = 0;
0069                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0070                 return
0071             end
0072         else
0073             if ~isequalwithequalnans(c1.(field), c2.(field))
0074                 result = 0;
0075                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0076                 return
0077             end
0078         end
0079     end
0080 end
0081 end
0082
```

[Home](#) > [m](#) > [helper](#) > ltpda_ri2fq.m

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

```
0001 function [f0, q]= ltpda_ri2fq(c)
0002
0003 % LTPDA_RI2FQ Convert complex pole/zero into frequency/Q pole/zero
representation.
0004 %
0005 % [f0, q]= ltpda_ri2fq(c)
0006 %
0007 % M Hewitson 26-01-07
0008 %
0009 % $Id: ltpda_ri2fq.m,v 1.4 2007/01/26 14:03:24 hewitson Exp $
0010 %
0011
0012
0013 if(nargin==0)
```



```
0014     disp('usage: [f0, q]= ltpda_ri2fq(c)');
0015     return
0016 end
0017
0018 a = real(c);
0019 b = imag(c);
0020
0021 f0 = -a*sqrt(b^2/a^2 + 1) / 2 / pi;
0022 q = sqrt(b^2/a^2 +1)/2;
0023
0024 % END
```

set

PURPOSE

SET sets a specwin property.

SYNOPSIS

function w = set(w, varargin)

DESCRIPTION

SET sets a specwin property.

%%%

DESCRIPTION: SET sets a specwin property.

CALL:

```
w = set(w, 'name',      'my window');
w = set(w, 'alpha',    -1);
w = set(w, 'psll',     -1);
w = set(w, 'rov',      -1);
w = set(w, 'nenbw',    -1);
w = set(w, 'w3db',     -1);
w = set(w, 'flatness', -1);
w = set(w, 'ws',       -1);
w = set(w, 'ws2',      -1);
w = set(w, 'win',      -1);
w = set(w, 'created',  time-object);
w = set(w, 'version',  'cvs version string');
```

VERSION: \$Id: set.m,v 1.4 2008/01/11 14:04:43 ingo Exp \$

HISTORY: 30-01-2007 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a specwin object into a string.

This function is called by:

- **plot** PLOT plots a specwin object.

SOURCE CODE

```

0001 function w = set(w, varargin)
0002 % SET sets a specwin property.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET sets a specwin property.
0007 %
0008 % CALL:          w = set(w, 'name',      'my window');
0009 %                w = set(w, 'alpha',    -1);
0010 %                w = set(w, 'psll',     -1);
0011 %                w = set(w, 'rov',      -1);
0012 %                w = set(w, 'nenbw',    -1);
0013 %                w = set(w, 'w3db',     -1);
0014 %                w = set(w, 'flatness', -1);
0015 %                w = set(w, 'ws',       -1);
0016 %                w = set(w, 'ws2',      -1);
0017 %                w = set(w, 'win',      -1);
0018 %                w = set(w, 'created',   time-object);
0019 %                w = set(w, 'version',   'cvs version string');
0020 %
0021 % VERSION:       $Id: set.m,v 1.4 2008/01/11 14:04:43 ingo Exp $
0022 %
0023 % HISTORY:       30-01-2007 M Hewitson
0024 %                Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0027
0028 VERSION = '$Id: set.m,v 1.4 2008/01/11 14:04:43 ingo Exp $';
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if isa(w, 'specwin') && ischar(varargin{1})
0033         in = char(varargin{1});
0034         if strcmp(in, 'Params')
0035             w = plist();
0036             return
0037         elseif strcmp(in, 'Version')
0038             w = VERSION;
0039             return
0040         end
0041     end
0042 end
0043
0044 property_argin = varargin;
0045
0046 fields = fieldnames(w);

```

```
0047
0048 while length(property_argin) >= 2
0049     prop = property_argin{1};
0050     val  = property_argin{2};
0051     property_argin = property_argin(3:end);
0052
0053     if ~ismember(prop, fields)
0054         error(['### ', prop, ' is not a valid specwin property.']);
0055     else
0056
0057         %% For all specwin objects (specwin can be a vector or a matrix)
0058         for ii = 1:length(w)
0059             w(ii).(prop) = val;
0060         end
0061
0062     end
0063
0064
0065 end
```

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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(c1,c2, 'ao/name')

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 1.5 2008/01/14 20:03:52 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:

- **eq** EQ overloads the == operator for time objects.
- **ne** NE overloads the ~= operator for time objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002 % EQ overloads the == operator for time objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EQ overloads the == operator for time objects.
0007 %
0008 %           All fields are checked.
0009 %
0010 % CALL:      result = eq(c1,c2)
0011 %           result = eq(c1,c2, exc_list)
0012 %           result = eq(c1,c2, 'property1', 'property2')
0013 %           result = eq(c1,c2, 'class/property', 'class/property')
0014 %
0015 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0016 %           result = eq(c1,c2, 'ao/name')
0017 %
0018 % INPUTS:    c1,c2      - input time objects
0019 %           exc_list - exception list
0020 %           List of properties which are not checked.
0021 %
0022 % OUTPUTS:   If the two time objects are considered equal, result == 1,
0023 %           otherwise, result == 0.
0024 %
0025 % VERSION:   $Id: eq.m,v 1.5 2008/01/14 20:03:52 ingo Exp $
0026 %
0027 % HISTORY:   29-08-2007 M Hewitson
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION = '$Id: eq.m,v 1.5 2008/01/14 20:03:52 ingo Exp $';
0033
0034 % Check if this is a call for parameters
0035 if nargin == 2
0036     if isa(c1, 'time') && ischar(c2)
0037         in = char(c2);
0038         if strcmp(in, 'Params')
0039             result = plist;
0040             return

```

```

0041     elseif strcmp(in, 'Version')
0042         result = VERSION;
0043         return
0044     end
0045 end
0046 end
0047
0048 result = 1;
0049
0050 %% Check class
0051 if ~strcmp(class(c1), class(c2))
0052     result = 0;
0053     return
0054 end
0055
0056 fields = fieldnames(c1);
0057
0058 for ii = 1:length(fields)
0059     field = fields{ii};
0060
0061     ck_field = {field, ['time/' field]};
0062
0063     %% Check tag
0064     if ~(any(ismember(ck_field, varargin)))
0065
0066         if isobject(c1.(field))
0067             if ~eq(c1.(field), c2.(field), varargin{:})
0068                 result = 0;
0069                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0070                 return
0071             end
0072         else
0073             if ~isequalwithequalnans(c1.(field), c2.(field))
0074                 result = 0;
0075                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0076                 return
0077             end
0078         end
0079     end
0080 end
0081 end

```

[Home](#) > [m](#) > [sigproc](#) > [frequency_domain](#) > ltpda_rms.m

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

```

0001 function varargout = ltpda_rms(varargin)
0002 % LTPDA_RMS Calculate RMS deviation from spectrum
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_RMS Calculate RMS deviation from spectrum
0007 %
0008 % CALL:          b=ltpda_rms(a)
0009 %
0010 % PARAMETERS:   b - analysis object containing RMS deviation
0011 %               a - input analysis object containing spectrum
0012 %
0013 % The following call returns a parameter list object that contains the
0014 % default parameter values:
0015 %
0016 % >> pl = ltpda_rms('Params')
0017 %
0018 % VERSION:      $Id: ltpda_rms.m,v 1.8 2007/11/08 10:35:04 hewitson Exp $
0019 %
0020 % NOTE:         Taken from code by: 1998.05.25      Masaki Ando
0021 %
0022 % HISTORY:      12-02-07 M Hewitson
0023 %               Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026
0027 % Check if this is a call for parameters
0028 if nargin == 1
0029     in = char(varargin{1});
0030     if strcmp(in, 'Params')
0031         varargout{1} = getDefaultPL();
0032         return
0033     end
0034 end
0035
0036 ALGONAME = mfilename;
0037 VERSION  = '$Id: ltpda_rms.m,v 1.8 2007/11/08 10:35:04 hewitson Exp $';
0038
0039
0040 % Get input analysis object
0041 a = varargin{1};
0042 if ~isa(a, 'ao')
0043     error('### first input argument should be an analysis object.');
```

```

0048 spe = [d.x d.y];
0049 f     = d.x;
0050
0051 % start and end frequencies
0052 s = f(1);
0053 e = f(end);
0054
0055 % compute integrated rms
0056 l1=spe(:,1)>=s;
0057 sp=spe(l1,:);
0058
0059 l2=sp(:,1)<=e;
0060 sp=sp(l2,:);
0061
0062 si=size(sp);
0063 li=si(1,1);
0064
0065 freq=sp(:,1);
0066 sp2=sp(:,2).^2;
0067 ms=sp2;
0068
0069 for i= li-1 :-1: 1
0070     ms(i)=ms(i+1)+(sp2(i+1)+sp2(i))*(freq(i+1)-freq(i))/2;
0071 end
0072
0073 % r=[freq,sqrt(ms)];
0074
0075 % Create output analysis object
0076 % create new output fsdata
0077 fs = fsdata(freq, sqrt(ms), d.fs);
0078 fs = set(fs, 'name', sprintf('RMS(%s)', d.name));
0079 fs = set(fs, 'yunits', d.yunits);
0080
0081 % create new output history
0082 h = history(ALGONAME, VERSION, [], a.hist);
0083
0084 % make output analysis object
0085 b = ao(fs, h);
0086
0087 % set name
0088 b = setnh(b, 'name', sprintf('RMS(%s)', a.name));
0089
0090 varargout{1} = b;
0091
0092
0093 %-----
0094 % Get default params
0095 function plo = getDefaultPL()
0096
0097 disp('* creating default plist...');
0098 plo = plist();
0099 disp('* done.');
```

0100

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@time](#) > set.m

set

PURPOSE

SET set time properties.

SYNOPSIS

function tt = set(tt, varargin)

DESCRIPTION

SET set time properties.

%%%

DESCRIPTION: SET set time properties.

CALL: tt = set(tt, 'property', 'value');
 tt = set(tt, 'property', 'value', 'direct');

EXAMPLES: tt = set(tt, 'name', 'my name');
 tt = set(tt, 'utc_epoch_milli', 1185468272023);
 tt = set(tt, 'timezone', 'UTC');
 tt = set(tt, 'timeformat', 'yyyy-mm-dd HH:MM:SS');
 tt = set(tt, 'timeformat', 31);
 tt = set(tt, 'timeformat', timeformat);
 tt = set(tt, 'time_str', '2007-07-26 18:44:32');
 tt = set(tt, 'created', '2007-07-26 18:44:32');
 tt = set(tt, 'version', 'cvs version string');

OPTION: 'direct' Sets the value direct without computing the new format

VERSION: \$Id: set.m,v 1.11 2008/01/15 17:36:36 ingo Exp \$

HISTORY: 23-07-2007 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:

- **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)

SOURCE CODE

```

0001 function tt = set(tt, varargin)
0002 % SET set time properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET set time properties.
0007 %
0008 % CALL:      tt = set(tt, 'property', 'value');
0009 %           tt = set(tt, 'property', 'value', 'direct');
0010 %
0011 % EXAMPLES:  tt = set(tt, 'name',          'my name');
0012 %           tt = set(tt, 'utc_epoch_milli', 1185468272023);
0013 %           tt = set(tt, 'timezone',        'UTC');
0014 %           tt = set(tt, 'timeformat',      'yyyy-mm-dd HH:MM:SS');
0015 %           tt = set(tt, 'timeformat',      31);
0016 %           tt = set(tt, 'timeformat',      timeformat);
0017 %           tt = set(tt, 'time_str',        '2007-07-26 18:44:32');
0018 %           tt = set(tt, 'created',         '2007-07-26 18:44:32');
0019 %           tt = set(tt, 'version',         'cvs version string');
0020 %
0021 % OPTION:    'direct' Sets the value direct without computing the new format
0022 %
0023 % VERSION:   $Id: set.m,v 1.11 2008/01/15 17:36:36 ingo Exp $
0024 %
0025 % HISTORY:   23-07-2007 Diepholz
0026 %           Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 VERSION = '$Id: set.m,v 1.11 2008/01/15 17:36:36 ingo Exp $';
0031
0032 % Check if this is a call for parameters or for the cvs-version number
0033 if nargin == 2
0034     if isa(tt, 'time') && ischar(varargin{1})
0035         in = char(varargin{1});

```

```

0036     if strcmp(in, 'Params')
0037         tt = plist();
0038         return
0039     elseif strcmp(in, 'Version')
0040         tt = VERSION;
0041         return
0042     end
0043 end
0044 end
0045
0046 if (nargout ~= 0)
0047
0048     compute = true;
0049     if nargin == 4 && strcmp(varargin{3}, 'direct')
0050         compute = false;
0051     end
0052
0053     fields = fieldnames(tt);
0054
0055     property_argin = varargin;
0056
0057     while length(property_argin) >= 2
0058         prop = property_argin{1};
0059         val = property_argin{2};
0060         property_argin = property_argin(3:end);
0061
0062         if ~ismember(prop, fields)
0063             error(['### ', prop, ' is not a valid time property.']);
0064         else
0065             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0066             % Set the timeformat AND transform the current time string to this format
0067             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0068             if (strcmp(prop, 'timeformat')) && (compute)
0069
0070                 if isa(val, 'timeformat')
0071                     tt.timeformat = val;
0072                     tt = set(tt, 'timeformat', char(val));
0073
0074                 else
0075                     t_format = java.text.SimpleDateFormat(tt.timeformat.default_java_str);
0076                     t_format.setTimeZone(tt.timezone);
0077                     current_time_str = char(t_format.format (tt.utc_epoch_milli));
0078
0079                     d_num = datenum (current_time_str, tt.timeformat.default_matlab_str);
0080                     tt.time_str = datestr(d_num, val);
0081
0082                     if isnumeric(val)
0083                         tt.timeformat = set(tt.timeformat, 'format_nr', val);
0084                     else
0085                         tt.timeformat = set(tt.timeformat, 'format_str', val);
0086                     end
0087                 end
0088             end
0089         end
0090     end
0091 end

```

```

0088
0089 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0090 % Set the timezone AND compute the current time string.
0091 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0092 elseif (strcmp(prop, 'timezone')) && (compute)
0093
0094     available_IDs = cell(java.util.TimeZone.getAvailableIDs);
0095
0096     if strcmp (class(val), 'sun.util.calendar.ZoneInfo')
0097         tt.timezone = val;
0098     elseif ischar(val) && ismember(val, available_IDs)
0099         tt.timezone = java.util.TimeZone.getTimeZone(val);
0100     else
0101         error (sprintf(['### Please use a timezone ID\n'...
0102                         'USE: get(time, 'timezone_IDs') OR a java
TimeZone']));
0103     end
0104
0105     t_format = java.text.SimpleDateFormat(tt.timeformat.default_java_str);
0106     t_format.setTimeZone(tt.timezone);
0107     tt.time_str = char(t_format.format(tt.utc_epoch_milli));
0108
0109 % Set the right time format
0110 if ~isempty(tt.timeformat.format_str)
0111     tt = set(tt, 'timeformat', tt.timeformat.format_str);
0112 elseif tt.timeformat.format_nr >= 0
0113     tt = set(tt, 'timeformat', tt.timeformat.format_nr);
0114 else
0115     error('### No time format is set. Should not happen.');
```

```

0116 end
0117
0118 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0119 % Set the time_str AND compute the utc_epoch_milli
0120 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0121 elseif (strcmp(prop, 'time_str')) && (compute)
0122
0123     if isempty(tt.timeformat.format_str)
0124         error (['### To set this property please set bevore a matlab format
string.'...
0125                 sprintf('\n') ...
0126                 '    USE: tt = set(tt, 'timeformat', 'yyyy-mm-dd HH:MM:
SS')']);
0127     end
0128
0129     parse_str = parse_time_string(val);
0130
0131     try
0132         d_num = datenum(val, parse_str);
0133 %         d_num = datenum(val, tt.timeformat.format_str);
0134         new_time_str = datestr(d_num, tt.timeformat.default_matlab_str);
0135     catch
0136         error('### The timeformat '%s' does not match to the value '%

```

```

s'', ...
0137         parse_str, val);
0138     end
0139
0140     % Compare the existing timeformat with the 'new' timeformat
0141     % Is it not the same then use the 'new' timeformat
0142     if strcmp (tt.timeformat.format_str, parse_str) == 0
0143         tt.timeformat = set(tt.timeformat, 'format_str', parse_str);
0144     end
0145
0146     t_format = java.text.SimpleDateFormat(tt.timeformat.default_java_str);
0147     t_format.setTimeZone(tt.timezone);
0148     new_time = t_format.parse(new_time_str);
0149
0150     tt.utc_epoch_milli = new_time.getTime;
0151     tt.time_str        = val;
0152
0153     % Set the right time format
0154     if ~isempty(tt.timeformat.format_str)
0155         tt = set(tt, 'timeformat', tt.timeformat.format_str);
0156     elseif tt.timeformat.format_nr >= 0
0157         tt = set(tt, 'timeformat', tt.timeformat.format_nr);
0158     else
0159         error('### No time format is set. Should not happen.');
```

%%%

```

0163     % Set the unix epoche time AND compute the time string
0164     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0165     elseif (strcmp(prop, 'utc_epoch_milli')) && (compute)
0166
0167         if ~isnumeric(val)
0168             error('### The utc epoch time must be a number');
0169         end
0170
0171         t_format = java.text.SimpleDateFormat(tt.timeformat.default_java_str);
0172         tt.time_str = char(t_format.format (val));
0173         tt.utc_epoch_milli = val;
0174
0175         % Set the right time format
0176         if ~isempty(tt.timeformat.format_str)
0177             tt = set(tt, 'timeformat', tt.timeformat.format_str);
0178         elseif tt.timeformat.format_nr >= 0
0179             tt = set(tt, 'timeformat', tt.timeformat.format_nr);
0180         else
0181             error('### No time format is set. Should not happen.');
```

%%%

```

0185     % set all other fields
0186     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0187     else
```



```

0188         tt.(prop) = val;
0189     end
0190 end
0191
0192 end
0193
0194 else
0195     if ischar(varargin{2})
0196         error('### please use: %s = set(%s, '%s'', '%s'');', ...
0197             inputname(1), ...
0198             inputname(1), ...
0199             varargin{1}, ...
0200             varargin{2});
0201     elseif isnumeric(varargin{2})
0202         error('### please use: %s = set(%s, '%s'', %d);', ...
0203             inputname(1), ...
0204             inputname(1), ...
0205             varargin{1}, ...
0206             varargin{2});
0207     else
0208         error('### please use: %s = set(%s, 'key'', 'value'');', ...
0209             inputname(1), ...
0210             inputname(1));
0211     end
0212 end
0213
0214 end
0215
0216 function ret = parse_time_string(in_str)
0217
0218     ret = in_str;
0219
0220     %           'match expression'      'MATLAB date format'
0221     parse = {'\d{2}:\d{2}:\d{2}', 'HH:MM:SS'; ...
0222             '\d{2}:\d{2}'         'MM:SS'; ...
0223             '\d{2}-\d{2}-\d{4}', 'dd-mm-yyyy'; ...
0224             '\d{4}-\d{2}-\d{2}', 'yyyy-mm-dd'; ...
0225             '\d{2}-\d{2}'         'mm-dd'; ...
0226             '\\.\d{1,3}',        '.FFF'};
0227
0228     for ii = 1:size(parse,1)
0229
0230         curr_parse = parse{ii,1};
0231         [match] = regexp(ret, curr_parse, 'match');
0232
0233         if ~isempty(match) && length(match) == 1
0234             replace = parse{ii,2};
0235             ret = strrep(ret, match{1}, replace);
0236         end
0237
0238     end
0239
0240 end

```

0241
0242
0243
0244
0245

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')

EXAMPLES:     result = eq(c1,c2, 'name', 'created')
               result = eq(c1,c2, 'ao/name')

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 1.3 2008/01/15 17:36:36 ingo 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:

- **eq** EQ overloads the == operator for timeformat objects.
- **ne** NE overloads the ~= operator for timeformat objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002 % EQ overloads the == operator for timeformat objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EQ overloads the == operator for timeformat objects.
0007 %
0008 %           All fields are checked.
0009 %
0010 % CALL:      result = eq(c1,c2)
0011 %           result = eq(c1,c2, exc_list)
0012 %           result = eq(c1,c2, 'property1', 'property2')
0013 %           result = eq(c1,c2, 'class/property', 'class/property')
0014 %
0015 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0016 %           result = eq(c1,c2, 'ao/name')
0017 %
0018 % INPUTS:    c1,c2      - input timeformat objects
0019 %           exc_list - exception list
0020 %           List of properties which are not checked.
0021 %
0022 % OUTPUTS:   If the two timeformat objects are considered equal, result == 1,
0023 %           otherwise, result == 0.
0024 %
0025 % VERSION:   $Id: eq.m,v 1.3 2008/01/15 17:36:36 ingo Exp $
0026 %
0027 % HISTORY:   29-08-2007 M Hewitson
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION = '$Id: eq.m,v 1.3 2008/01/15 17:36:36 ingo Exp $';
0033
0034 % Check if this is a call for parameters or for the cvs-version number
0035 if nargin == 2
0036     if isa(c1, 'timeformat') && ischar(c2)
0037         in = char(c2);
0038         if strcmp(in, 'Params')
0039             result = plist();
0040             return
0041         elseif strcmp(in, 'Version')

```

```

0042     result = VERSION;
0043     return
0044 end
0045 end
0046 end
0047
0048 result = 1;
0049
0050 %% Check class
0051 if ~strcmp(class(c1), class(c2))
0052     result = 0;
0053     return
0054 end
0055
0056 fields = fieldnames(c1);
0057
0058 for ii = 1:length(fields)
0059     field = fields{ii};
0060
0061     ck_field = {field, ['timeformat/' field]};
0062
0063     %% Check tag
0064     if ~(any(ismember(ck_field, varargin)))
0065
0066         if isobject(c1.(field))
0067             if ~eq(c1.(field), c2.(field), varargin{:})
0068                 result = 0;
0069                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0070                 return
0071             end
0072         else
0073             if ~isequalwithhequalnans(c1.(field), c2.(field))
0074                 result = 0;
0075                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0076                 return
0077             end
0078         end
0079
0080     end
0081 end
0082
0083
0084 % function result = eq(tf1,tf2)
0085 %
0086 % % EQ overloads the == operator for timeformat objects.
0087 % %
0088 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0089 % %
0090 % % DESCRIPTION: EQ overloads the == operator for timeformat objects.
0091 % %
0092 % %         Each field is checked.
0093 % %

```

```
0094 % % CALL:      result = eq(tf1,tf2)
0095 % %
0096 % % INPUTS:    tf1,tf2 - input plist objects
0097 % %
0098 % % OUTPUTS:   If the two timeformat objects are considered equal, result ==
0099 % %             1,
0100 % %             otherwise, result == 0.
0101 % % VERSION:   $Id: eq.m,v 1.3 2008/01/15 17:36:36 ingo Exp $
0102 % %
0103 % % HISTORY:    29-08-2007 M Hewitson
0104 % %             Creation
0105 % %
0106 % %%%%%%%%%%%%%%%
0107 %
0108 % % Check each field
0109 % result = isequal(struct(tf1), struct(tf2));
0110 %
0111 %
```

[Home](#) > [m](#) > [helper](#) > ltpda_saveobj.m

ltpda_saveobj

PURPOSE

SAVEOBJ saves an object to a file.

SYNOPSIS

function 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 A0)
 pl - parameter list with a 'filename' parameter

OUTPUTS:

VERSION: \$Id: ltpda_saveobj.m,v 1.1 2007/08/29 06:29:51 hewitson Exp \$

HISTORY: 28-08-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function saveobj(a, pl)
0002
0003 % SAVEOBJ saves an object to a file.
0004 %
```

```

0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SAVEOBJ saves an object to a file.
0008 %
0009 % CALL:      saveobj(obj, pl)
0010 %
0011 % INPUTS:    obj - an object (for example, an A0)
0012 %            pl  - parameter list with a 'filename' parameter
0013 %
0014 % OUTPUTS:
0015 %
0016 % VERSION: $Id: ltpda_saveobj.m,v 1.1 2007/08/29 06:29:51 hewitson Exp $
0017 %
0018 % HISTORY: 28-08-2007 M Hewitson
0019 %           Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023
0024 % get filename
0025 filename = find(pl, 'filename');
0026
0027 % Inspect filename
0028 [path,name,ext,vers] = fileparts(filename);
0029
0030 switch ext
0031     case '.mat'
0032
0033         save(filename, 'a');
0034
0035     case '.xml'
0036
0037         % convert object to xml
0038         x = xml(a);
0039         % Save the XML document.
0040         xmlwrite(filename,x.docNode);
0041
0042     otherwise
0043         error('### unknown file extension.');
```


[Home](#) > [classes](#) > [@timeformat](#) > set.m

set

PURPOSE

SET set timeformat properties.

SYNOPSIS

function tf = set(tf, varargin)

DESCRIPTION

SET set timeformat properties.

%%%

DESCRIPTION: SET set timeformat properties.

CALL: tf = set(tf, 'property', 'value');
 tf = set(tf, 'property', 'value', 'direct');

EXAMPLES: tt = set(tt, 'format_str', 'yyyy.dd.mm HH:MM:SS AM');
 tt = set(tt, 'format_nr', 0);
 tt = set(tt, 'default_matlab_str', 'yyyy-mm-dd HH:MM:SS');
 tt = set(tt, 'default_matlab_nr', 31);
 tt = set(tt, 'default_java_str', 'yyyy-MM-dd HH:mm:ss');

VERSION: \$Id: set.m,v 1.3 2008/01/15 17:36:36 ingo 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:

SOURCE CODE

```

0001 function tf = set(tf, varargin)
0002 % SET set timeformat properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET set timeformat properties.
0007 %
0008 % CALL:          tf = set(tf, 'property', 'value');
0009 %               tf = set(tf, 'property', 'value', 'direct');
0010 %
0011 % EXAMPLES:      tt = set(tt, 'format_str',      'yyyy.dd.mm HH:MM:SS AM');
0012 %               tt = set(tt, 'format_nr',        0);
0013 %               tt = set(tt, 'default_matlab_str', 'yyyy-mm-dd HH:MM:SS');
0014 %               tt = set(tt, 'default_matlab_nr', 31);
0015 %               tt = set(tt, 'default_java_str',  'yyyy-MM-dd HH:mm:ss');
0016 %
0017 % VERSION:       $Id: set.m,v 1.3 2008/01/15 17:36:36 ingo Exp $
0018 %
0019 % HISTORY:       23-07-2007 Diepholz
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: set.m,v 1.3 2008/01/15 17:36:36 ingo Exp $';
0025
0026 % Check if this is a call for parameters
0027 if nargin == 2
0028     if isa(tf, 'timeformat') && ischar(varargin{1})
0029         in = char(varargin{1});
0030         if strcmp(in, 'Params')
0031             tf = plist();
0032             return
0033         elseif strcmp(in, 'Version')
0034             tf = VERSION;
0035             return
0036         end
0037     end
0038 end
0039
0040 if (nargout ~= 0)
0041
0042     compute = true;
0043     if nargin == 4 && strcmp(varargin{3}, 'direct')
0044         compute = false;
0045     end
0046
0047     property_argin = varargin;
0048
0049     fields = fieldnames(tf);
0050
0051     while length(property_argin) >= 2
0052         prop = property_argin{1};

```

```

0053     val = property_argin{2};
0054     property_argin = property_argin(3:end);
0055
0056     if ~ismember(prop, fields)
0057         error(['### ', prop, ' is not a valid time property.']);
0058     else
0059
0060         % Set the format string
0061         if (strcmp(prop, 'format_str')) && (compute)
0062             if ischar(val)
0063                 tf.format_str = val;
0064                 if strcmp(tf.format_str, tf.default_matlab_str)
0065                     tf.format_nr = tf.default_matlab_nr;
0066                 else
0067                     tf.format_nr = -1;
0068                 end
0069             else
0070                 error('### Please use for %s a format-string', tf.(field));
0071             end
0072
0073             % Set the format number
0074             elseif (strcmp(prop, 'format_nr')) && (compute)
0075                 if isnumeric(val)
0076                     tf.format_nr = val;
0077                     if (tf.format_nr == tf.default_matlab_nr)
0078                         tf.format_str = tf.default_matlab_str;
0079                     else
0080                         tf.format_str = '';
0081                     end
0082                 else
0083                     error('### Please use for %s a matlab format-number', tf.(field));
0084                 end
0085             elseif (strcmp(prop, {'default_matlab_str', ...
0086                                   'default_matlab_nr', ...
0087                                   'default_java_str'}))
0088                 error('### Do not set one of the default values. Only the time
0089 constructor should set this values. ');
0090             else
0091                 tf.(prop) = val;
0092             end
0093         end
0094
0095     end
0096 end
0097
0098 else
0099     if ischar(varargin{2})
0100         error('### please use: %s = set(%s, '%s'', '%s'');', ...
0101               inputname(1), ...
0102               inputname(1), ...
0103               varargin{1}, ...

```

```
0104         varargin{2});
0105 elseif isnumeric(varargin{2})
0106     error('### please use: %s = set(%s, '%s'', %d);', ...
0107         inputname(1), ...
0108         inputname(1), ...
0109         varargin{1}, ...
0110         varargin{2});
0111 else
0112     error('### please use: %s = set(%s, 'key'', 'value'');', ...
0113         inputname(1), ...
0114         inputname(1));
0115 end
0116 end
```

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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')

result = eq(c1,c2, 'ao/name')

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 1.2 2007/10/22 11:51:52 ingo Exp \$

HISTORY: 29-08-2007 M Hewitson

Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- **eq** EQ overloads the == operator for timespan objects.

This function is called by:

- **eq** EQ overloads the == operator for timespan objects.
- **ne** NE overloads the ~= operator for timespan objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002
0003 % EQ overloads the == operator for timespan objects.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: EQ overloads the == operator for timespan objects.
0008 %
0009 %           All fields are checked.
0010 %
0011 % CALL:      result = eq(c1,c2)
0012 %           result = eq(c1,c2,  exc_list)
0013 %           result = eq(c1,c2, 'property1', 'property2')
0014 %           result = eq(c1,c2, 'class/property', 'class/property')
0015 %
0016 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0017 %           result = eq(c1,c2, 'ao/name')
0018 %
0019 % INPUTS:    c1,c2      - input timespan objects
0020 %           exc_list - exception list
0021 %           List of properties which are not checked.
0022 %
0023 % OUTPUTS:   If the two timespan objects are considered equal, result == 1,
0024 %           otherwise, result == 0.
0025 %
0026 % VERSION:   $Id: eq.m,v 1.2 2007/10/22 11:51:52 ingo Exp $
0027 %
0028 % HISTORY:   29-08-2007 M Hewitson
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032
0033 result = 1;
0034
0035 %% Check class
0036 if ~strcmp(class(c1), class(c2))
0037     result = 0;
0038     return
0039 end
0040
0041 fields = fieldnames(c1);
0042

```

```

0043 for ii = 1:length(fields)
0044     field    = fields{ii};
0045
0046     ck_field = {field, ['timespan/' field]};
0047
0048     %% Check tag
0049     if ~(any(ismember(ck_field, varargin)))
0050
0051         if isobject(c1.(field))
0052             if ~eq(c1.(field), c2.(field), varargin{:})
0053                 result = 0;
0054                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0055                 return
0056             end
0057         else
0058             if ~isequalwithhequalnans(c1.(field), c2.(field))
0059                 result = 0;
0060                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0061                 return
0062             end
0063         end
0064
0065     end
0066 end
0067
0068
0069 % function result = eq(t1,t2)
0070 %
0071 % % EQ overloads the == operator for timespan objects.
0072 % %
0073 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0074 % %
0075 % % DESCRIPTION: EQ overloads the == operator for timespan objects.
0076 % %
0077 % %             Start and stop times are compared.
0078 % %
0079 % % CALL:       result = eq(ts1,ts2)
0080 % %
0081 % % INPUTS:     ts1,ts2 - input timespan objects
0082 % %
0083 % % OUTPUTS:    If the two timespan objects are considered equal, result == 1,
0084 % %             otherwise, result == 0.
0085 % %
0086 % % VERSION:    $Id: eq.m,v 1.2 2007/10/22 11:51:52 ingo Exp $
0087 % %
0088 % % HISTORY:    29-08-2007 M Hewitson
0089 % %             Creation
0090 % %
0091 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0092 %
0093 % % Assume true

```

```
0094 % result = 1;
0095 %
0096 % %% Check start
0097 % if t1.start ~= t2.start
0098 %     result = 0;
0099 %     return
0100 % end
0101 %
0102 % %% Check end
0103 % if t1.end ~= t2.end
0104 %     result = 0;
0105 %     return
0106 % end
0107 %
0108 % %% Check timeformat
0109 % if t1.timeformat ~= t2.timeformat
0110 %     result = 0;
0111 %     return
0112 % end
0113 %
0114 % %% Check timezone
0115 % if t1.timezone ~= t2.timezone
0116 %     result = 0;
0117 %     return
0118 % end
```


[Home](#) > [m](#) > [etc](#) > ltpda_startup.m

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 1.20 2008/01/10 19:53:29 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function ltpda_startup
0002
0003 % This is the startup file for ltpda. It should be run once in the MATLAB
0004 % session before using any features of ltpda. The best way to ensure this
0005 % is to create a file called startup.m and put this somewhere in your
0006 % MATLAB path. In this file you should have the command 'ltpda_startup'.
0007 %
0008 % M Hewitson 16-03-07
0009 %
0010 % $Id: ltpda_startup.m,v 1.20 2008/01/10 19:53:29 hewitson Exp $
0011 %
0012
0013 %-----
0014 % User Configurable parameters
0015 %
0016
```



```

0069 for j=1:length(logo)
0070     disp([strpad(sprintf('|      %s ', char(logo{j})), ll-1) '|']);
0071 end
0072 disp([strpad('|', ll-1) '|'])
0073 disp([strpad(sprintf('|      Welcome to %s', v.Name), ll-1) '|'])
0074 disp([strpad('|', ll-1) '|'])
0075 disp([strpad(sprintf('|      Version: %s', v.Version), ll-1) '|'])
0076 disp([strpad(sprintf('|      Release: %s', v.Release), ll-1) '|'])
0077 disp([strpad(sprintf('|      Date: %s', v.Date), ll-1) '|'])
0078 disp([strpad('|', ll-1) '|'])
0079 disp(l1);
0080
0081 %-----
0082 % set page properties for printing
0083 if strcmp(upper(USE_LTPDA_PRINT_SETTINGS), 'YES')
0084     set(0, 'DefaultFigurePaperOrientation', 'portrait');
0085     set(0, 'DefaultFigurePaperType', 'A4');
0086     set(0, 'DefaultFigurePaperUnits', 'inches');
0087 end
0088
0089 %-----
0090 % Plot settings
0091 if strcmp(upper(USE_LTPDA_PLOT_SETTINGS), 'YES')
0092     set(0, 'DefaultAxesFontSize', 14);
0093     set(0, 'DefaultAxesFontWeight', 'bold');
0094     set(0, 'DefaultAxesLineWidth', 1);
0095     set(0, 'DefaultLineLineWidth', 2);
0096     set(0, 'defaultlinemarkersize', 10);
0097     set(0, 'DefaultAxesXColor', [0.2 0.2 0.2]);
0098     set(0, 'DefaultAxesYColor', [0.2 0.2 0.2]);
0099     set(0, 'DefaultAxesGridLineStyle', '-');
0100     set(0, 'DefaultAxesMinorGridLineStyle', '-');
0101     set(0, 'defaultfigurenumbertitle', 'on');
0102     set(0, 'DefaultFigureColor', 'w');
0103     set(0, 'DefaultFigurePosition', [100 100 800 600]);
0104     set(0, 'DefaultFigurePaperPositionMode', 'auto')
0105
0106 end
0107
0108 %-----
0109 % format of numbers on MATLAB terminal
0110 format long g
0111
0112 % -----
0113 % MySQL LTPDA Repository Server Settings
0114 %
0115 if exist('REPOUSER', 'var') == 0
0116     REPOUSER = '';
0117 end
0118 if exist('REPOPASSWD', 'var') == 0
0119     REPOPASSWD = '';
0120 end
0121 setappdata(0, 'ltpda_repo_user', REPOUSER);

```

```

0122 setappdata(0, 'ltpda_repo_pass', REPOPASSWD);
0123 setappdata(0, 'mysql_driver', DBDRIVER);
0124
0125 setappdata(0, 'ltpda_repo_gui_fontsize', REPO_GUI_FONTSIZE);
0126 setappdata(0, 'ltpda_passwd_win_fontsize', PASSWD_WIN_FONTSIZE);
0127
0128 % add mysql JDBC driver to the java path
0129 mysqldriver = [fileparts(which('ltpda_startup')) '../mysql/mysql-connector-
java-5.0.6-bin.jar'];
0130 javaaddpath(mysqldriver);
0131
0132
0133 setdbprefs('NullStringRead', 'Null');
0134 setdbprefs('NullNumberRead', 'NaN');
0135
0136 % -----
0137 % General Variables
0138 setappdata(0, 'wrapstringat', WRAP_STRINGS); % wrap strings at this length for
0139 % history plot nodes
0140 setappdata(0, 'wraplegendstringat', WRAP_LEGEND_STRINGS); % wrap strings at
this length for legends
0141 setappdata(0, 'xmlsetsize', XML_SET_SIZE); % Max size of an xml data set <Set></
Set>
0142
0143 setappdata(0, 'ltpda_default_plot_colors', {[0.8 0.2 0.2], ...
0144 [0.2 0.2 0.8], ...
0145 [0.2 0.9 0.2], ...
0146 [0.37 0.9 0.83], ...
0147 [0.888 0.163 0.9], ...
0148 [0 0 0], ...
0149 [0 207 255]/255, ...
0150 [255 128 0]/255, ...
0151 [143 0 0]/255, ...
0152 [255 207 0]/255, ...
0153 [0.9 0.266 0.593]});
0154
0155 setappdata(0, 'timezone', TIMEZONE);
0156 setappdata(0, 'time_format_str', TIME_FORMAT_STR);
0157
0158
0159 % -----
0160 % Version Variables
0161
0162 vs = ver;
0163 for j=1:length(vs)
0164     v = vs(j);
0165     switch v.Name
0166         case 'MATLAB'
0167             matlab_version = [v.Version ' ' v.Release];
0168         case 'Signal Processing Toolbox'
0169             sigproc_version = [v.Version ' ' v.Release];
0170         case 'LTPDA Toolbox'
0171             ltpda_version = [v.Version ' ' v.Release];

```

```
0172     end
0173 end
0174
0175 setappdata(0, 'matlab_version', matlab_version);
0176 setappdata(0, 'sigproc_version', sigproc_version);
0177 setappdata(0, 'ltpda_version', ltpda_version);
0178
0179 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@timespan](#) > set.m

set

PURPOSE

SET set timespan properties.

SYNOPSIS

function ts = set(ts, varargin)

DESCRIPTION

SET set timespan properties.

%%%

DESCRIPTION: SET set timespan properties.

CALL: ts = set(ts, 'property', 'value');

EXAMPLES: ts = set(ts, 'name', 'my name');
 ts = set(ts, 'start', '14:00:00');
 ts = set(ts, 'end', '14:00:05');
 ts = set(ts, 'timeformat', 'HH:MM:SS');
 ts = set(ts, 'timeformat', 31);
 ts = set(ts, 'timezone', 'UTC');
 tt = set(ts, 'created', time-object);
 tt = set(ts, 'version', 'cvs version string');

VERSION: \$Id: set.m,v 1.8 2007/10/16 17:37:21 ingo Exp \$

HISTORY: 23-07-2007 Diepholz
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a timespan object into a string.
- [set](#) SET set timespan properties.

This function is called by:

- **set** SET set timespan properties.
- **timespan** TIMESPAN timespan object class constructor.

SOURCE CODE

```

0001 function ts = set(ts, varargin)
0002 % SET set timespan properties.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET set timespan properties.
0007 %
0008 % CALL:      ts = set(ts, 'property', 'value');
0009 %
0010 % EXAMPLES:  ts = set(ts, 'name',      'my name');
0011 %            ts = set(ts, 'start',     '14:00:00');
0012 %            ts = set(ts, 'end',       '14:00:05');
0013 %            ts = set(ts, 'timeformat', 'HH:MM:SS');
0014 %            ts = set(ts, 'timeformat', 31);
0015 %            ts = set(ts, 'timezone',  'UTC');
0016 %            tt = set(ts, 'created',    time-object);
0017 %            tt = set(ts, 'version',    'cvs version string');
0018 %
0019 % VERSION:    $Id: set.m,v 1.8 2007/10/16 17:37:21 ingo Exp $
0020 %
0021 % HISTORY:    23-07-2007 Diepholz
0022 %              Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 VERSION = '$Id: set.m,v 1.8 2007/10/16 17:37:21 ingo Exp $';
0027
0028 % Check if this is a call for parameters or for the cvs-version number
0029 if nargin == 2
0030     if isa(ts, 'timespan') && ischar(varargin{1})
0031         in = char(varargin{1});
0032         if strcmp(in, 'Params')
0033             ts = plist();
0034             return
0035         elseif strcmp(in, 'Version')
0036             ts = VERSION;
0037             return
0038         end
0039     end
0040 end
0041
0042 if (nargout ~= 0)
0043
0044     fields = fieldnames(ts);
0045
0046     property_argin = varargin;
0047

```

```

0048 while length(property_argin) >= 2
0049     prop = property_argin{1};
0050     val = property_argin{2};
0051     property_argin = property_argin(3:end);
0052
0053     if ~ismember(prop, fields)
0054         error(['### ', prop, ' is not a valid timespan property.']);
0055     else
0056         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0057         % Set the start time.
0058         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0059         if strcmp(prop, 'start')
0060
0061             if isa(val, 'time')
0062                 val = set(val, 'timeformat', ts.timeformat);
0063                 ts.start = val;
0064             elseif ischar(val)
0065                 ts.start = time(val, char(ts.timeformat));
0066             end
0067
0068             % Compute the interval
0069             ts = compute_interval(ts);
0070
0071             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072             %
0073             % Set the end time.
0074             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0075             elseif strcmp(prop, 'end')
0076
0077                 if isa(val, 'time')
0078                     val = set(val, 'timeformat', ts.timeformat);
0079                     ts.end = val;
0080                 elseif ischar(val)
0081                     ts.end = time(val, char(ts.timeformat));
0082                 end
0083
0084                 % Compute the interval
0085                 ts = compute_interval(ts);
0086
0087                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0088                 % Set the timeformat.
0089                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0090             elseif strcmp(prop, 'timeformat')
0091
0092                 if isa(val, 'timeformat')
0093                     ts.timeformat = val;
0094
0095                     ts.start = set(ts.start, 'timeformat', char(val));
0096                     ts.end = set(ts.end, 'timeformat', char(val));
0097                 elseif ischar(val) || isnumeric(val)
0098                     ts.start = set(ts.start, 'timeformat', val);

```



```

0099         ts.end = set(ts.end, 'timeformat', val);
0100
0101         ts.timeformat = ts.start.timeformat;
0102     end
0103
0104     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0105     % Set the timezone
0106     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0107     elseif strcmp(prop, 'timezone')
0108
0109         ts.start = set(ts.start, 'timezone', val);
0110         ts.end = set(ts.end, 'timezone', val);
0111
0112         ts.timezone = ts.start.timezone;
0113
0114     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0115     % set all other fields.
0116     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0117     else
0118         ts.(prop) = val;
0119     end
0120 end
0121
0122 end
0123
0124 else
0125     if ischar(varargin{2})
0126         error('### please use: %s = set(%s, '%s'', '%s'');', ...
0127             inputname(1), ...
0128             inputname(1), ...
0129             varargin{1}, ...
0130             varargin{2});
0131     elseif isnumeric(varargin{2})
0132         error('### please use: %s = set(%s, '%s'', %d);', ...
0133             inputname(1), ...
0134             inputname(1), ...
0135             varargin{1}, ...
0136             varargin{2});
0137     else
0138         error('### please use: %s = set(%s, 'key'', 'value'');', ...
0139             inputname(1), ...
0140             inputname(1));
0141     end
0142 end

```

[Home](#) > [classes](#) > [@provenance](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a provenance from a provenance xml file.

SYNOPSIS

function ps = xmlparse(p, xmlfile)

DESCRIPTION

XMLPARSE creates a provenance from a provenance xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a provenance object into a string.

This function is called by:

- [provenance](#) PROVENANCE constructors for provenance class.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a provenance from a provenance xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```

```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'provenance')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```

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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(c1,c2, 'ao/name')

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 1.7 2007/11/27 12:47:19 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:

- **eq** EQ overloads the == operator for tsdata objects.
- **ne** NE overloads the ~= operator for tsdata objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002
0003 % EQ overloads the == operator for tsdata objects.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: EQ overloads the == operator for tsdata objects.
0008 %
0009 %           All fields are checked.
0010 %
0011 % CALL:      result = eq(c1,c2)
0012 %           result = eq(c1,c2,  exc_list)
0013 %           result = eq(c1,c2, 'property1', 'property2')
0014 %           result = eq(c1,c2, 'class/property', 'class/property')
0015 %
0016 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0017 %           result = eq(c1,c2, 'ao/name')
0018 %
0019 % INPUTS:    c1,c2      - input tsdata objects
0020 %           exc_list - exception list
0021 %                   List of properties which are not checked.
0022 %
0023 % OUTPUTS:   If the two tsdata objects are considered equal, result == 1,
0024 %           otherwise, result == 0.
0025 %
0026 % VERSION:   $Id: eq.m,v 1.7 2007/11/27 12:47:19 ingo Exp $
0027 %
0028 % HISTORY:   29-08-2007 M Hewitson
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032
0033 VERSION = '$Id: eq.m,v 1.7 2007/11/27 12:47:19 ingo Exp $';
0034
0035 % Check if this is a call for parameters
0036 if nargin == 2
0037     if isa(c1, 'tsdata') && ischar(c2)
0038         in = char(c2);
0039         if strcmp(in, 'Params')
0040             result = plist;

```

```

0041     return
0042     elseif strcmp(in, 'Version')
0043         result = VERSION;
0044         return
0045     end
0046 end
0047 end
0048
0049 result = 1;
0050
0051 %% Check class
0052 if ~strcmp(class(c1), class(c2))
0053     result = 0;
0054     return
0055 end
0056
0057 fields = fieldnames(c1);
0058
0059 for ii = 1:length(fields)
0060     field = fields{ii};
0061
0062     ck_field = {field, ['tsdata/' field]};
0063
0064     %% Check tag
0065     if ~(any(ismember(ck_field, varargin)))
0066
0067         if isobject(c1.(field))
0068             if ~eq(c1.(field), c2.(field), varargin{:})
0069                 result = 0;
0070                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0071                 return
0072             end
0073         else
0074             if ~isequalwithhequalnans(c1.(field), c2.(field))
0075                 result = 0;
0076                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0077                 return
0078             end
0079         end
0080
0081     end
0082 end
0083
0084

```

[Home](#) > [m](#) > [helper](#) > ltpda_str2cells.m

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

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```

0001 function newCell = ltpda_str2cells(someString)
0002 % LTPDA_STR2CELLS Take a single string and separate out individual "elements"
0003 % into a new cell array.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_STR2CELLS Take a single string and separate out individual
0007 %               "elements" into a new cell array. Elements are defined as non-
0008 %               blank characters separated by
0009 %               spaces.
0010 %               Similar to str2cell, except str2cell requires an array of
0011 %               strings.
0012 %               str2cells requires only 1 string.
0013 % CALL:         newCell = ltpda_str2cells(aString)
0014 %
0015 % INPUTS:       aString - string
0016 %
0017 % OUTPUTS:      newCell - cell array of strings
0018 %
0019 % EXAMPLE:      Consider the following string in the workspace:
0020 %
0021 % aString = ' a   b           c d  efgh ij       klmnopqrs t u v w xyz   '
0022 % newCell = 'a'
0023 %           'b'
0024 %           'c'
0025 %           'd'
0026 %           'efgh'
0027 %           'ij'
0028 %           'klmnopqrs'
0029 %           't'

```



```

0030 %          'u'
0031 %          'v'
0032 %          'w'
0033 %          'xyz'
0034 %
0035 % REMARK:  This is copied from a file found on MathWorks File Exchange.
0036 %
0037 % VERSION: $Id: ltpda_str2cells.m,v 1.7 2007/07/13 12:17:39 ingo Exp $
0038 %
0039 % HISTORY: 26-01-2007 M Hewitson
0040 %          Creation
0041 %
0042 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0043
0044 % If someString is empty then return an empty Cell
0045 if isempty(someString)
0046     newCell = {};
0047     return
0048 end
0049
0050 % Trim off any leading & trailing blanks
0051 someString=strtrim(someString);
0052
0053 % Locate all the white-spaces
0054 spaces=isspace(someString);
0055
0056 % Build the cell array
0057 idx=0;
0058 while sum(spaces)~=0
0059     idx=idx+1;
0060     newCell{idx}=strtrim(someString(1:find(spaces==1,1,'first')));
0061     someString=strtrim(someString(find(spaces==1,1,'first')+1:end));
0062     spaces=isspace(someString);
0063 end
0064 newCell{idx+1}=someString;
0065

```

[Home](#) > [classes](#) > [@tsdata](#) > set.m

set

PURPOSE

SET sets a tsdata property.

SYNOPSIS

function ts = set(ts, varargin)

DESCRIPTION

SET sets a tsdata property.

%%%

DESCRIPTION: SET sets a tsdata property.

```
CALL: ts = set(ts, 'name',      'my_name');
      ts = set(ts, 'fs',        fs);
      ts = set(ts, 'x',         x);
      ts = set(ts, 'y',         y);
      ts = set(ts, 'nsecs',     nsecs);
      ts = set(ts, 't0',        Matlab serial date number);
      ts = set(ts, 'xunits',    'Time');
      ts = set(ts, 'yunits',    'Volts');
      ts = set(ts, 'created',   time());
      ts = set(ts, 'version',   'tsdata/set.m,v 1.7...');
```

VERSION: \$Id: set.m,v 1.12 2007/11/13 16:03:20 ingo Exp \$

HISTORY: 30-01-07 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

```

0001 function ts = set(ts, varargin)
0002 % SET sets a tsdata property.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET sets a tsdata property.
0007 %
0008 % CALL: ts = set(ts, 'name',      'my_name');
0009 %         ts = set(ts, 'fs',      fs);
0010 %         ts = set(ts, 'x',      x);
0011 %         ts = set(ts, 'y',      y);
0012 %         ts = set(ts, 'nsecs',   nsecs);
0013 %         ts = set(ts, 't0',      Matlab serial date number);
0014 %         ts = set(ts, 'xunits',  'Time');
0015 %         ts = set(ts, 'yunits',  'Volts');
0016 %         ts = set(ts, 'created', time());
0017 %         ts = set(ts, 'version', 'tsdata/set.m,v 1.7...');
0018 %
0019 % VERSION: $Id: set.m,v 1.12 2007/11/13 16:03:20 ingo Exp $
0020 %
0021 % HISTORY: 30-01-07 M Hewitson
0022 %           Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 VERSION = '$Id: set.m,v 1.12 2007/11/13 16:03:20 ingo Exp $';
0027
0028 % Check if this is a call for parameters
0029 if nargin == 2
0030     if isa(ts, 'tsdata') && ischar(varargin{1})
0031         in = char(varargin{1});
0032         if strcmp(in, 'Params')
0033             ts = plist;
0034             return
0035         elseif strcmp(in, 'Version')
0036             ts = VERSION;
0037             return
0038         end
0039     end
0040 end
0041
0042 if (nargout ~= 0)
0043
0044     propArgin = varargin;
0045
0046     while length(propArgin) >= 2
0047         prop = propArgin{1};
0048         val  = propArgin{2};
0049         propArgin = propArgin(3:end);

```

```

0050     switch prop
0051         case 'name'
0052             ts.name = val;
0053         case 'created'
0054             ts.created = val;
0055         case 'version'
0056             ts.version = val;
0057         case 'x'
0058             ts.x = val;
0059         case 'y'
0060             ts.y = val;
0061         case 'fs'
0062             ts.fs = val;
0063         case 'nsecs'
0064             ts.nsecs = val;
0065         case 'xunits'
0066             ts.xunits = val;
0067         case 'yunits'
0068             ts.yunits = val;
0069         case 't0'
0070             if isa(val, 'time')
0071                 ts.t0 = val;
0072             elseif ischar(val) || isnumeric(val)
0073                 ts.t0 = time(val);
0074             end
0075         otherwise
0076             error([val, ' is not a property of tsdata class']);
0077     end
0078 end
0079
0080 else
0081     if ischar(varargin{2})
0082         error('### please use: %s = set(%s, '%s'', '%s'');', ...
0083             inputname(1), ...
0084             inputname(1), ...
0085             varargin{1}, ...
0086             varargin{2});
0087     elseif isnumeric(varargin{2})
0088         error('### please use: %s = set(%s, '%s'', %d);', ...
0089             inputname(1), ...
0090             inputname(1), ...
0091             varargin{1}, ...
0092             varargin{2});
0093     else
0094         error('### please use: %s = set(%s, 'key'', 'value'');', ...
0095             inputname(1), ...
0096             inputname(1));
0097     end
0098 end

```

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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(c1,c2, 'ao/name')

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 1.6 2007/11/27 13:07:28 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:

- **eq** EQ overloads the == operator for xydata objects.
- **ne** NE overloads the ~= operator for xydata objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002
0003 % EQ overloads the == operator for xydata objects.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: EQ overloads the == operator for xydata objects.
0008 %
0009 %           All fields are checked.
0010 %
0011 % CALL:      result = eq(c1,c2)
0012 %           result = eq(c1,c2,  exc_list)
0013 %           result = eq(c1,c2, 'property1', 'property2')
0014 %           result = eq(c1,c2, 'class/property', 'class/property')
0015 %
0016 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0017 %           result = eq(c1,c2, 'ao/name')
0018 %
0019 % INPUTS:    c1,c2      - input xydata objects
0020 %           exc_list - exception list
0021 %                   List of properties which are not checked.
0022 %
0023 % OUTPUTS:   If the two xydata objects are considered equal, result == 1,
0024 %           otherwise, result == 0.
0025 %
0026 % VERSION:   $Id: eq.m,v 1.6 2007/11/27 13:07:28 ingo Exp $
0027 %
0028 % HISTORY:   29-08-2007 M Hewitson
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032
0033 VERSION = '$Id: eq.m,v 1.6 2007/11/27 13:07:28 ingo Exp $';
0034
0035 % Check if this is a call for parameters
0036 if nargin == 2
0037     if isa(c1, 'xydata') && ischar(c2)
0038         in = char(c2);
0039         if strcmp(in, 'Params')
0040             result = plist;
0041             return

```

```

0042     elseif strcmp(in, 'Version')
0043         result = VERSION;
0044         return
0045     end
0046 end
0047 end
0048
0049 result = 1;
0050
0051 %% Check class
0052 if ~strcmp(class(c1), class(c2))
0053     result = 0;
0054     return
0055 end
0056
0057 fields = fieldnames(c1);
0058
0059 for ii = 1:length(fields)
0060     field = fields{ii};
0061
0062     ck_field = {field, ['xydata/' field]};
0063
0064     %% Check tag
0065     if ~(any(ismember(ck_field, varargin)))
0066
0067         if isobject(c1.(field))
0068             if ~eq(c1.(field), c2.(field), varargin{:})
0069                 result = 0;
0070                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0071                 return
0072             end
0073         else
0074             if ~isequalwithhequalnans(c1.(field), c2.(field))
0075                 result = 0;
0076                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0077                 return
0078             end
0079         end
0080
0081     end
0082 end
0083
0084
0085 % function result = eq(c1,c2, varargin)
0086 %
0087 % % EQ overloads the == operator for xydata objects.
0088 % %
0089 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0090 % %
0091 % % DESCRIPTION: EQ overloads the == operator for xydata objects.
0092 % %
0093 % % The 'name', 'xunits', 'yunits, and data values are checked.

```

```

0094 % %
0095 % % CALL:      result = eq(c1,c2)
0096 % %
0097 % % INPUTS:     c1,c2 - input xydata objects
0098 % %
0099 % % OUTPUTS:    If the two xydata objects are considered equal, result == 1,
0100 % %              otherwise, result == 0.
0101 % %
0102 % % VERSION:    $Id: eq.m,v 1.6 2007/11/27 13:07:28 ingo Exp $
0103 % %
0104 % % HISTORY:     29-08-2007 M Hewitson
0105 % %              Creation
0106 % %
0107 % %%%%%%%%%%%%%%
%
0108 %
0109 % % Assume equality to begin with
0110 % result = 1;
0111 %
0112 % %% Check name
0113 % if ~ismember('name', varargin)
0114 %     if ~strcmp(c1.name, c2.name)
0115 %         result = 0;
0116 %         return
0117 %     end
0118 % end
0119 %
0120 % %% check xunits
0121 % if ~ismember('xunits', varargin)
0122 %     if ~strcmp(c1.xunits, c2.xunits)
0123 %         result = 0;
0124 %         return
0125 %     end
0126 % end
0127 %
0128 % %% check yunits
0129 % if ~ismember('yunits', varargin)
0130 %     if ~strcmp(c1.yunits, c2.yunits)
0131 %         result = 0;
0132 %         return
0133 %     end
0134 % end
0135 %
0136 % %% Check x data
0137 % if ~ismember('x', varargin)
0138 %     if length(c1.x) ~= length(c2.x)
0139 %         result = 0;
0140 %         return
0141 %     end
0142 %
0143 %     if c1.x ~= c2.x
0144 %         result = 0;
0145 %         return

```



```
0146 % end
0147 % end
0148 %
0149 % %% Check y data
0150 % if ~ismember('y', varargin)
0151 %     if length(c1.y) ~= length(c2.y)
0152 %         result = 0;
0153 %         return
0154 %     end
0155 %
0156 %     if c1.y ~= c2.y
0157 %         result = 0;
0158 %         return
0159 %     end
0160 % end
```

[Home](#) > [m](#) > [helper](#) > ltpda_strs2cells.m

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

```
0001 function cell = ltpda_strs2cells(varargin)
0002 % LTPDA_STRS2CELLS convert a set of input strings to a cell array.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_STRS2CELLS convert a set of input strings to a cell array.
0007 %
0008 % CALL:          cell = ltpda_strs2cells(str1, str2, ...)
0009 %
0010 % INPUTS:        str1 - string
0011 %                str2 - string
0012 %                ...
0013 %
0014 % OUTPUTS:       cell - cell array of the input strings
0015 %
0016 % VERSION: $Id: ltpda_strs2cells.m,v 1.5 2007/07/13 12:17:39 ingo Exp $
0017 %
0018 % HISTORY: 26-01-2007 M Hewitson
0019 %           Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 cell = [];
0024 for j=1:nargin
0025     cell = [cell cellstr(varargin{j})];
0026
0027
0028 end
0029
0030
0031
0032 % END

```

[Home](#) > [classes](#) > [@xydata](#) > set.m

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 1.7 2007/11/13 16:03:20 ingo Exp \$

HISTORY: 30-01-07 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

```

0001 function xy = set(xy, varargin)
0002 % SET sets a xydata property.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET sets a xydata property.
0007 %
0008 % CALL: c = set(c, 'name',      'my_name');
0009 %       c = set(c, 'x',        X);
0010 %       c = set(c, 'x',        [1 2 3]);
0011 %       c = set(c, 'y',        Y);
0012 %       c = set(c, 'y',        [1 2 3]);
0013 %       c = set(c, 'xunits',   'sec');
0014 %       c = set(c, 'yunits',   'V');
0015 %       c = set(c, 'created',   time());
0016 %       c = set(c, 'version',   'xydata/set.m,v 1.7...');
0017 %
0018 % VERSION: $Id: set.m,v 1.7 2007/11/13 16:03:20 ingo Exp $
0019 %
0020 % HISTORY: 30-01-07 M Hewitson
0021 %           Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 VERSION = '$Id: set.m,v 1.7 2007/11/13 16:03:20 ingo Exp $';
0026
0027 % Check if this is a call for parameters
0028 if nargin == 2
0029     if isa(xy, 'xydata') && ischar(varargin{1})
0030         in = char(varargin{1});
0031         if strcmp(in, 'Params')
0032             xy = plist;
0033             return
0034         elseif strcmp(in, 'Version')
0035             xy = VERSION;
0036             return
0037         end
0038     end
0039 end
0040
0041 if (nargout ~= 0)
0042
0043     propArgin = varargin;
0044
0045     while length(propArgin) >= 2
0046         prop = propArgin{1};
0047         val = propArgin{2};
0048         propArgin = propArgin(3:end);
0049         switch prop
0050             case 'name'
0051                 xy.name = val;
0052             case 'created'

```

```

0053     xy.created = val;
0054     case 'version'
0055         xy.version = val;
0056     case 'y'
0057         xy.y = val;
0058     case 'x'
0059         xy.x = val;
0060     case 'xunits'
0061         xy.xunits = val;
0062     case 'yunits'
0063         xy.yunits = val;
0064     otherwise
0065         error([val, ' is not a property of xydata class']);
0066     end
0067 end
0068
0069 else
0070     if ischar(varargin{2})
0071         error('### please use: %s = set(%s, '%s'', '%s'');', ...
0072             inputname(1), ...
0073             inputname(1), ...
0074             varargin{1}, ...
0075             varargin{2});
0076     elseif isnumeric(varargin{2})
0077         error('### please use: %s = set(%s, '%s'', %d);', ...
0078             inputname(1), ...
0079             inputname(1), ...
0080             varargin{1}, ...
0081             varargin{2});
0082     else
0083         error('### please use: %s = set(%s, 'key'', 'value'');', ...
0084             inputname(1), ...
0085             inputname(1));
0086     end
0087 end

```

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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')

result = eq(c1,c2, 'ao/name')

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 1.1 2007/12/24 19:47:06 hewitson 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:

- **eq** EQ overloads the == operator for xyzdata objects.
- **ne** NE overloads the ~= operator for xyzdata objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002
0003 % EQ overloads the == operator for xyzdata objects.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: EQ overloads the == operator for xyzdata objects.
0008 %
0009 %           All fields are checked.
0010 %
0011 % CALL:      result = eq(c1,c2)
0012 %           result = eq(c1,c2,  exc_list)
0013 %           result = eq(c1,c2, 'property1', 'property2')
0014 %           result = eq(c1,c2, 'class/property', 'class/property')
0015 %
0016 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0017 %           result = eq(c1,c2, 'ao/name')
0018 %
0019 % INPUTS:    c1,c2      - input xyzdata objects
0020 %           exc_list - exception list
0021 %           List of properties which are not checked.
0022 %
0023 % OUTPUTS:   If the two xyzdata objects are considered equal, result == 1,
0024 %           otherwise, result == 0.
0025 %
0026 % VERSION:   $Id: eq.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $
0027 %
0028 % HISTORY:   24-12-2007 M Hewitson
0029 %           Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032
0033 VERSION = '$Id: eq.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $';
0034
0035 % Check if this is a call for parameters
0036 if nargin == 2
0037     if ischar(c2)
0038         in = char(c2);
0039         if strcmp(in, 'Params')
0040             result = plist;
0041             return
0042         elseif strcmp(in, 'Version')

```



```

0043     result = VERSION;
0044     return
0045 end
0046 end
0047 end
0048
0049 result = 1;
0050
0051 %% Check class
0052 if ~strcmp(class(c1), class(c2))
0053     result = 0;
0054     return
0055 end
0056
0057 fields = fieldnames(c1);
0058
0059 for ii = 1:length(fields)
0060     field = fields{ii};
0061
0062     ck_field = {field, ['xyzdata/' field]};
0063
0064     %% Check tag
0065     if ~(any(ismember(ck_field, varargin)))
0066
0067         if isobject(c1.(field))
0068             if ~eq(c1.(field), c2.(field), varargin{:})
0069                 result = 0;
0070                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0071                 return
0072             end
0073         else
0074             if ~isequalwithequalnans(c1.(field), c2.(field))
0075                 result = 0;
0076                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0077                 return
0078             end
0079         end
0080
0081     end
0082 end
0083
0084
0085 % function result = eq(c1,c2, varargin)
0086 %
0087 % % EQ overloads the == operator for xydata objects.
0088 % %
0089 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0090 % %
0091 % % DESCRIPTION: EQ overloads the == operator for xydata objects.
0092 % %
0093 % %             The 'name', 'xunits', 'yunits, and data values are checked.
0094 % %

```

```

0095 % % CALL:      result = eq(c1,c2)
0096 % %
0097 % % INPUTS:    c1,c2 - input xydata objects
0098 % %
0099 % % OUTPUTS:    If the two xydata objects are considered equal, result == 1,
0100 % %              otherwise, result == 0.
0101 % %
0102 % % VERSION:    $Id: eq.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $
0103 % %
0104 % % HISTORY:     29-08-2007 M Hewitson
0105 % %              Creation
0106 % %
0107 % %%%%%%%%%%%%%%
%
0108 %
0109 % % Assume equality to begin with
0110 % result = 1;
0111 %
0112 % %% Check name
0113 % if ~ismember('name', varargin)
0114 %     if ~strcmp(c1.name, c2.name)
0115 %         result = 0;
0116 %         return
0117 %     end
0118 % end
0119 %
0120 % %% check xunits
0121 % if ~ismember('xunits', varargin)
0122 %     if ~strcmp(c1.xunits, c2.xunits)
0123 %         result = 0;
0124 %         return
0125 %     end
0126 % end
0127 %
0128 % %% check yunits
0129 % if ~ismember('yunits', varargin)
0130 %     if ~strcmp(c1.yunits, c2.yunits)
0131 %         result = 0;
0132 %         return
0133 %     end
0134 % end
0135 %
0136 % %% Check x data
0137 % if ~ismember('x', varargin)
0138 %     if length(c1.x) ~= length(c2.x)
0139 %         result = 0;
0140 %         return
0141 %     end
0142 %
0143 %     if c1.x ~= c2.x
0144 %         result = 0;
0145 %         return
0146 %     end

```

```
0147 % end
0148 %
0149 % %% Check y data
0150 % if ~ismember('y', varargin)
0151 %     if length(c1.y) ~= length(c2.y)
0152 %         result = 0;
0153 %         return
0154 %     end
0155 %
0156 %     if c1.y ~= c2.y
0157 %         result = 0;
0158 %         return
0159 %     end
0160 % end
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [helper](#) > ltpda_structcat.m

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

```
0001 function out = ltpda_structcat(varargin)
0002 % LTPDA_STRUCTCAT concatonate structures to make one large structure.
0003 %
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_STRUCTCAT concatonate structures to make one large
0007 %               structure.
0008 %
0009 % CALL:         out = ltpda_structcat(struct1, struct2, ...)
0010 %
0011 % INPUTS:       struct1 - a structure
0012 %               struct2 - a structure
0013 %
0014 % OUTPUTS:      out - structure with all fields of input structures.
0015 %
0016 % VERSION: $Id: ltpda_structcat.m,v 1.5 2007/07/13 12:17:39 ingo Exp $
0017 %
0018 % HISTORY: 26-01-2007 M Hewitson
0019 %           Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 out = [];
0024
0025 for s=1:nargin
0026
0027     % get first input structure
0028     st = varargin{s};
0029
0030     % check fields
0031     fields = fieldnames(st);
0032     nf     = length(fields);
0033
0034     for f=1:nf
0035
0036         field = char(fields(f));
0037         % check if this field already exists
0038         if isfield(out, field)
0039             warning(sprintf('!!! duplicate field '%s' found - skipping.', field));
0040         else
0041             % we can add this field
0042             eval(sprintf('out.%s = st.%s;', field, field));
0043         end
0044     end
0045 end
0046
0047 % END

```

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 1.1 2007/12/24 19:47:06 hewitson Exp \$

HISTORY: 24-12-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```

0001 function xyz = set(xyz, varargin)
0002 % SET sets an xyzdata property.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET sets an xyzdata property.
0007 %
0008 % CALL: c = set(c, 'name',      'my_name');
0009 %         c = set(c, 'x',        X);
0010 %         c = set(c, 'x',        [1 2 3]);
0011 %         c = set(c, 'y',        Y);
0012 %         c = set(c, 'y',        [1 2 3]);
0013 %         c = set(c, 'z',        Z);
0014 %         c = set(c, 'z',        [1 2 3]);
0015 %         c = set(c, 'xunits',    's');
0016 %         c = set(c, 'yunits',    'Hz');
0017 %         c = set(c, 'zunits',    'V');
0018 %         c = set(c, 'created',    time());
0019 %         c = set(c, 'version',    'xyzdata/set.m,v 1.7...');
0020 %
0021 % VERSION: $Id: set.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $
0022 %
0023 % HISTORY: 24-12-07 M Hewitson
0024 %           Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0027
0028 VERSION = '$Id: set.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $';
0029
0030 % Check if this is a call for parameters
0031 if nargin == 2
0032     if ischar(varargin{2})
0033         in = char(varargin{2});
0034         if strcmp(in, 'Params')
0035             xyz = plist;
0036             return
0037         elseif strcmp(in, 'Version')
0038             xyz = VERSION;
0039             return
0040         end
0041     end
0042 end
0043
0044 if (nargout ~= 0)
0045
0046     propArgin = varargin;
0047
0048     while length(propArgin) >= 2
0049         prop = propArgin{1};
0050         val = propArgin{2};
0051         propArgin = propArgin(3:end);

```

```

0052     switch prop
0053         case 'name'
0054             xyz.name = val;
0055         case 'created'
0056             xyz.created = val;
0057         case 'version'
0058             xyz.version = val;
0059         case 'z'
0060             xyz.z = val;
0061         case 'y'
0062             xyz.y = val;
0063         case 'x'
0064             xyz.x = val;
0065         case 'xunits'
0066             xyz.xunits = val;
0067         case 'yunits'
0068             xyz.yunits = val;
0069         case 'zunits'
0070             xyz.zunits = val;
0071         otherwise
0072             error([val, ' is not a property of xyzdata class']);
0073     end
0074 end
0075
0076 else
0077     if ischar(varargin{2})
0078         error('### please use: %s = set(%s, '%s'', '%s'');', ...
0079             inputname(1), ...
0080             inputname(1), ...
0081             varargin{1}, ...
0082             varargin{2});
0083     elseif isnumeric(varargin{2})
0084         error('### please use: %s = set(%s, '%s'', %d);', ...
0085             inputname(1), ...
0086             inputname(1), ...
0087             varargin{1}, ...
0088             varargin{2});
0089     else
0090         error('### please use: %s = set(%s, 'key'', 'value'');', ...
0091             inputname(1), ...
0092             inputname(1));
0093     end
0094 end

```


[Home](#) > [classes](#) > [@time](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a time from a time xml file.

SYNOPSIS

function ps = xmlparse(p, xmlfile)

DESCRIPTION

XMLPARSE creates a time from a time xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a time object into a string.

This function is called by:

- [time](#) TIME time object class constructor.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a time from a time xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```

```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'time')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```

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')

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(c1,c2, 'ao/name')

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 1.3 2008/01/02 17:58:51 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:

- **eq** EQ overloads the == operator for zero objects.
- **ne** NE overloads the ~= operator for zero objects.

SOURCE CODE

```

0001 function result = eq(c1,c2, varargin)
0002 % EQ overloads the == operator for zero objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EQ overloads the == operator for zero objects.
0007 %
0008 %           All fields are checked.
0009 %
0010 % CALL:      result = eq(c1,c2)
0011 %           result = eq(c1,c2, exc_list)
0012 %           result = eq(c1,c2, 'property1', 'property2')
0013 %           result = eq(c1,c2, 'class/property', 'class/property')
0014 %
0015 % EXAMPLES:  result = eq(c1,c2, 'name', 'created')
0016 %           result = eq(c1,c2, 'ao/name')
0017 %
0018 % INPUTS:    c1,c2      - input zero objects
0019 %           exc_list - exception list
0020 %           List of properties which are not checked.
0021 %
0022 % OUTPUTS:   If the two zero objects are considered equal, result == 1,
0023 %           otherwise, result == 0.
0024 %
0025 % VERSION:   $Id: eq.m,v 1.3 2008/01/02 17:58:51 ingo Exp $
0026 %
0027 % HISTORY:   29-08-2007 M Hewitson
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION    = '$Id: eq.m,v 1.3 2008/01/02 17:58:51 ingo Exp $';
0033
0034 % Check if this is a call for parameters
0035 if nargin == 2
0036     if isa(c1, 'zero') && ischar(c2)
0037         in = char(c2);
0038         if strcmp(in, 'Params')
0039             result = plist;
0040             return

```

```

0041     elseif strcmp(in, 'Version')
0042         result = VERSION;
0043         return
0044     end
0045 end
0046 end
0047
0048 result = 1;
0049
0050 %% Check class
0051 if ~strcmp(class(c1), class(c2))
0052     result = 0;
0053     return
0054 end
0055
0056 fields = fieldnames(c1);
0057
0058 for ii = 1:length(fields)
0059     field = fields{ii};
0060
0061     ck_field = {field, ['zero/' field]};
0062
0063     %% Check tag
0064     if ~(any(ismember(ck_field, varargin)))
0065
0066         if isobject(c1.(field))
0067
0068             if length(c1.(field)) ~= length(c2.(field))
0069                 result = 0;
0070                 return
0071             end
0072
0073             for jj = 1:length(c1.(field))
0074                 if ~eq(c1.(field)(jj), c2.(field)(jj), varargin{:})
0075                     result = 0;
0076                     disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0077                     return
0078                 end
0079             end
0080         else
0081             if ~isequalwithhequalnans(c1.(field), c2.(field))
0082                 result = 0;
0083                 disp(sprintf('\nNOT EQUAL: %s.%s', class(c1), field));
0084                 return
0085             end
0086         end
0087     end
0088 end
0089 end
0090

```


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 explicitelty 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

```

0001 function sout = ltpda_structidx(sin, idx, varargin)
0002 % LTPDA_STRUCTIDX Index a structure.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_STRUCTIDX Index a structure. All fields are assumed
0007 %               to be vectors of the same length, except those explicitly
0008 %               ignored.
0009 %
0010 %               You can give an additional list of field names which will be
0011 %               ignored in the indexing.
0012 % CALL:         sout = ltpda_structidx(sin, idx)
0013 %               = ltpda_structidx(sin, idx, 'field1', 'field2')
0014 %
0015 % INPUTS:       sin      - a struct
0016 %               idx      - index
0017 %               field1   - field name
0018 %               field2   - field name
0019 %               ...
0020 %
0021 % OUTPUTS:      sout    - a struct
0022 %
0023 % VERSION: $Id: ltpda_structidx.m,v 1.5 2007/07/13 12:17:39 ingo Exp $
0024 %
0025 % HISTORY: 26-01-2007 M Hewitson
0026 %           Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 Nigf = nargin-2;
0031
0032 % examine fields
0033 fields = fieldnames(sin);
0034 nf      = length(fields);
0035
0036 for f=1:nf
0037     fn = char(fields(f));
0038     skip = 0;
0039     for j=1:Nigf
0040
0041         if strcmp(fn, char(varargin{j}))
0042             skip = 1;
0043             str = sprintf('sout.%s = sin.%s;', fn, fn);
0044             eval(str);
0045         end
0046     end
0047 end
0048

```



```
0049     if ~skip
0050         str = sprintf('sout.%s = sin.%s(idx);', fn, fn);
0051         eval(str);
0052     end
0053 end
0054
0055
0056
0057
0058
0059 % END
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@zero](#) > set.m

set

PURPOSE

SET set a zero property.

SYNOPSIS

function p = set(p, varargin)

DESCRIPTION

SET set a zero property.

%%%

DESCRIPTION: SET a history property.

CALL: z = set(z, 'name', 'name');
 z = set(z, 'f', f);
 z = set(z, 'q', q);
 z = set(z, 'ri', ri);
 z = set(z, 'version', 'plist/set.m,v 1.7...');
 z = set(z, 'created', time-object);

VERSION: \$Id: set.m,v 1.7 2008/01/03 15:33:49 ingo Exp \$

HISTORY: 30-01-07 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

```
0001 function p = set(p, varargin)
0002 % SET set a zero property.
```

```

0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET a history property.
0007 %
0008 % CALL:          z = set(z, 'name',      'name');
0009 %                z = set(z, 'f',         f);
0010 %                z = set(z, 'q',         q);
0011 %                z = set(z, 'ri',        ri);
0012 %                z = set(z, 'version', 'plist/set.m,v 1.7...');
0013 %                z = set(z, 'created',   time-object);
0014 %
0015 % VERSION:       $Id: set.m,v 1.7 2008/01/03 15:33:49 ingo Exp $
0016 %
0017 % HISTORY:       30-01-07 M Hewitson
0018 %                 Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION = '$Id: set.m,v 1.7 2008/01/03 15:33:49 ingo Exp $';
0023
0024 % Check if this is a call for parameters or for the cvs-version number
0025 if nargin == 2
0026     if isa(p, 'zero') && ischar(varargin{1})
0027         in = char(varargin{1});
0028         if strcmp(in, 'Params')
0029             p = plist();
0030             return
0031         elseif strcmp(in, 'Version')
0032             p = VERSION;
0033             return
0034         end
0035     end
0036 end
0037
0038 if (nargout ~= 0)
0039
0040     propArgin = varargin;
0041     while length(propArgin) >= 2
0042         prop = varargin{1};
0043         val = varargin{2};
0044         propArgin = propArgin(3:end);
0045         switch prop
0046             case 'name'
0047                 p.name = val;
0048             case 'version'
0049                 p.version = val;
0050             case 'created'
0051                 p.created = val;
0052             case 'f'
0053                 p.f = val;
0054                 p.ri = pfq2ri(p.f, p.q);

```

```

0055     case 'q'
0056         p.q = val;
0057         p.ri = pfq2ri(p.f, p.q);
0058     case 'ri'
0059         if length(val) == 1
0060             p.ri = [val conj(val)];
0061         else
0062             p.ri = val;
0063         end
0064         [p.f, p.q] = pri2fq(p.ri);
0065     otherwise
0066         error([val, ' is not a property of zero class']);
0067     end
0068 end
0069
0070 else
0071     if ischar(varargin{2})
0072         error('### please use: %s = set(%s, '%s'', '%s'');', ...
0073             inputname(1), ...
0074             inputname(1), ...
0075             varargin{1}, ...
0076             varargin{2});
0077     elseif isnumeric(varargin{2})
0078         error('### please use: %s = set(%s, '%s'', %d);', ...
0079             inputname(1), ...
0080             inputname(1), ...
0081             varargin{1}, ...
0082             varargin{2});
0083     else
0084         error('### please use: %s = set(%s, key, 'value'');', ...
0085             inputname(1), ...
0086             inputname(1));
0087     end
0088 end

```

[Home](#) > [classes](#) > [@timeformat](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a timeformat from a timeformat xml file.

SYNOPSIS

function ps = xmlparse(p, xmlfile)

DESCRIPTION

XMLPARSE creates a timeformat from a timeformat xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a timeformat object into a string.

This function is called by:

- [timeformat](#) TIMEFORMAT timeformat object class constructor.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a timeformat from a timeformat xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```

```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'timeformat')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```

[Home](#) > [m](#) > [sigproc](#) > [frequency_domain](#) > ltpda_tfe.m

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')
```

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- `ltpda_xspect` LTPDA_XSPEC performs cross-spectral analysis of various forms.

This function is called by:

SUBFUNCTIONS

- function `plo = getDefaultPL()`

SOURCE CODE

```
0001 function varargout = ltpda_tfe(varargin)
0002 % LTPDA_TFE makes transfer function estimates of the time-series objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_TFE makes transfer function estimates of the time-series
0007 %               objects in the input analysis objects. Transfer functions are
0008 %               computed using MATLAB's tfestimate (>> help tfestimate).
0009 %
0010 % CALL:        b = ltpda_tfe(a1,a2,a3,...,pl)
0011 %
0012 % INPUTS:      b      - output analysis objects
0013 %              aN     - input analysis objects (at least two)
0014 %              pl     - input parameter list
0015 %
0016 % OUTPUTS:     The output matrix b contains NxN transfer functions estimates (all
0017 %               possible pairs of input AOs).
0018 %
0019 %               If the last input argument is a parameter list (plist) it is used.
0020 %               The following parameters are recognised.
0021 %
0022 % PARAMETERS:  'Win'   - a specwin window object [default: Kaiser -200dB psll]
0023 %               'Nolap' - segment overlap (number of samples) [default: taken
0024 %               from window function]
0025 %               'Nfft'  - number of samples in each fft [default: fs of input
0026 %               data]
0027 %
0028 % VERSION:     $Id: ltpda_tfe.m,v 1.14 2007/07/16 12:52:21 ingo Exp $
0029 %
0030 % HISTORY:     07-02-2007 M Hewitson
0031 %               Creation
0032 %
0033 % The following call returns a parameter list object that contains the
0034 % default parameter values:
```



```

0033 %
0034 % >> pl = ltpda_tfe('Params')
0035 %
0036 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0037
0038 ALGONAME = mfilename;
0039 VERSION  = '$Id: ltpda_tfe.m,v 1.14 2007/07/16 12:52:21 ingo Exp $';
0040
0041 % Check if this is a call for parameters
0042 if nargin == 1
0043     in = char(varargin{1});
0044     if strcmp(in, 'Params')
0045         varargout{1} = getDefaultPL();
0046         return
0047     end
0048 end
0049
0050 % capture input variable names
0051 invars = {};
0052 as      = [];
0053 ps      = [];
0054 for j=1:nargin
0055     invars = [invars cellstr(inputname(j))];
0056     if isa(varargin{j}, 'ao')
0057         as = [as varargin{j}];
0058     end
0059     if isa(varargin{j}, 'plist')
0060         ps = [ps varargin{j}];
0061     end
0062 end
0063
0064
0065 % check plist
0066 if isempty(ps)
0067     pl = getDefaultPL();
0068 else
0069     pl = combine(ps, getDefaultPL);
0070 end
0071
0072 varargout{1} = ltpda_xspec(as, pl, 'TF', ALGONAME, VERSION, invars);
0073
0074
0075
0076
0077
0078 %-----
0079 % Get default params
0080 function plo = getDefaultPL()
0081
0082 disp('* creating default plist...');
0083 plo = plist();
0084 plo = append(plo, param('Nfft', -1));

```

```
0085 plo = append(plo, param('Win', specwin('Kaiser', 1, 100)));  
0086 plo = append(plo, param('Nolap', -1));  
0087 disp('* done.');
```

```
0088
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@fsdata](#) > setFreq.m

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 1.4 2007/11/30 19:36:05 ingo 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

```
0001 function fsd = setFreq(fsd, varargin)
0002 % SETFREQ sets the frequency vector of a fsdata object based on the length of
the y vector and the sample rate.
```

```

0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION:  SETFREQ sets the frequency vector of a fsdata object based on the
0007 %                length of the y vector and the sample rate.
0008 %
0009 % CALL:          fsd = setFreq(fsd)
0010 %
0011 % VERSION:       $Id: setFreq.m,v 1.4 2007/11/30 19:36:05 ingo Exp $
0012 %
0013 % HISTORY:       31-01-2007 M Hewitson
0014 %                Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: setFreq.m,v 1.4 2007/11/30 19:36:05 ingo Exp $';
0019
0020 if nargin == 2 && isa(fsd, 'fsdata') && ischar(varargin{1})
0021     if strcmp(varargin{1}, 'Params')
0022         fsd = plist();
0023         return
0024     elseif strcmp(varargin{1}, 'Version')
0025         fsd = VERSION;
0026         return
0027     end
0028 end
0029
0030 fsd.x = linspace(0, fsd.fs/2, length(fsd.y));
0031 fsd = reshapeF(fsd);
0032
0033 % END

```

[Home](#) > [classes](#) > [@timespan](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a timespan from a timespan xml file.

SYNOPSIS

function ps = xmlparse(p, xmlfile)

DESCRIPTION

XMLPARSE creates a timespan from a timespan xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a timespan object into a string.

This function is called by:

- [timespan](#) TIMESPAN timespan object class constructor.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a timespan from a timespan xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```

```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'timespan')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```

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 through the history tree
              to see only a part of this tree. If the user will use the plot
              function the plot will be printed in a new figure.

CALL:        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 1.13 2007/12/18 14:00:09 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] = getNodevalue(cNode, s)
- function c_str = split_by_comma(str, pref)

SOURCE CODE

```

0001 function explore_ao(varargin)
0002 % EXPLORE_AO for exploring analysis objects and plotting/displaying their fields
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EXPLORE_AO for exploring analysis objects and plotting/displaying
0007 %               their fields. It is possible to browse through the history tree
0008 %               to see only a part of this tree. If the user will use the plot
0009 %               function the plot will be printed in a new figure.
0010 %
0011 % CALL:        explore_ao;                % Read the ao's from the 'base' workspace
0012 %               explore_ao(ao);
0013 %               explore_ao(ao_vector);
0014 %               explore_ao(ao_matrix);
0015 %
0016 % HINT:        The idea and the core source code are taken from:
0017 %               Hassan Lahdili (hassan.lahdili@crc.ca)
0018 %               Communications Research Centre (CRC) | Advanced Audio Systems (AAS)
0019 %               www.crc.ca | www.crc.ca/aas
0020 %               Ottawa. Canada
0021 %               CRC Advanced Audio Systems - Ottawa 16/02/2005 2004-2005
0022 %
0023 % VERSION: $Id: explore_ao.m,v 1.13 2007/12/18 14:00:09 ingo Exp $
0024 %
0025 % HISTORY: 10-06-07 Diepholz
0026 %               Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029 %
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```



```

0032 %           Define the Positions           %
0033 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0034
0035 % figure position
0036 FIG_X   = .150;
0037 FIG_Y   = .150;
0038 FIG_dX  = .700;
0039 FIG_dY  = .600;
0040
0041 % tree position
0042 TREE_X   = .000;
0043 TREE_Y   = .000;
0044 TREE_dX  = .350;
0045 TREE_dY  = 1.000;
0046
0047 % plot field position
0048 PLOT_X   = .370;
0049 PLOT_Y   = .105;
0050 PLOT_dX  = .610;
0051 PLOT_dY  = .720;
0052
0053 % display field position
0054 DISP_X   = TREE_X;
0055 DISP_Y   = .750;
0056 DISP_dX  = TREE_dX;
0057 DISP_dY  = 1-DISP_Y;
0058
0059 % explorer name position
0060 EXPL_NAME_dX = .370;
0061 EXPL_NAME_dY = .045;
0062 EXPL_NAME_X  = PLOT_X+(PLOT_dX-EXPL_NAME_dX)/2;
0063 EXPL_NAME_Y  = .030;
0064 % EXPL_NAME_Y = .065;
0065
0066 % info fields position
0067 N_INFOS = 3;
0068
0069 INFO_X   = PLOT_X;
0070 INFO_Y   = .915;
0071 INFO_dX  = PLOT_dX / N_INFOS;
0072 INFO_dY  = .034;
0073
0074 BG_COLOR = [.925 .914 .847];
0075
0076 MAX_HIST_DEPTH = 4;
0077
0078 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0079 %           Check the input           %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081
0082 % Get AO list from workspace
0083 vars = evalin('base','who');
0084 aoList = [];

```

```

0085 for j=1:length(vars)
0086     var = evalin('base', vars{j});
0087     if isa(var, 'ao')
0088         % add to list
0089         aoList = [aoList var];
0090     end
0091 end
0092
0093 obj      = [];
0094 obj_value = {};
0095 obj_name  = {};
0096
0097 % Read the ao's from the 'base' workspace
0098
0099 if nargin == 0
0100     tree_name = 'obj_ws';
0101     ws_vars   = evalin('base', 'whos');
0102
0103 elseif nargin == 1
0104     tree_name = inputname(1);
0105     tmp       = varargin{1};
0106
0107     if iscell(tmp)
0108         ws_vars = [];
0109         for jj = 1:length(tmp)
0110             ws_vars(jj).name = [inputname(1) '{' num2str(jj) '}'];
0111             ws_vars(jj).class = class(tmp{jj});
0112         end
0113
0114     else
0115         ws_vars = whos('tmp');
0116         ws_vars.name = inputname(1);
0117     end
0118
0119 else
0120     error('#####');
0121 end
0122
0123 for ii=1:length(ws_vars)
0124
0125     obj = evalin('base', ws_vars(ii).name);
0126
0127     if isobject(obj)
0128
0129         % the object in the workspace is a single value
0130         if numel(obj) == 1
0131             obj_value{end+1} = obj;
0132             obj_name{end+1} = [ws_vars(ii).class ':' ws_vars(ii).name];
0133         else
0134
0135             [n,m] = size(obj);
0136
0137             % the ao in the workspace is a vector

```

```

0138         if n == 1 || m == 1
0139
0140             for jj=1:length(obj)
0141                 obj_value{end+1} = obj(jj);
0142                 obj_name{end+1} = [ws_vars(ii).class ':' ws_vars(ii).name '(' num2str
(jj) ')'];
0143             end
0144
0145             % the ao in the workspace is a matrix
0146         elseif n > 1 && m > 1
0147
0148             for gg = 1:n
0149                 for hh = 1:m
0150                     obj_value{end+1} = obj(gg, hh);
0151                     obj_name{end+1} = [ws_vars(ii).class ':' ws_vars(ii).name '('
num2str(gg) ',' num2str(hh) ')'];
0152                 end
0153             end
0154
0155         else
0156             error ('### this should not happen. ');
0157         end
0158
0159     end
0160
0161 end
0162 end
0163
0164
0165 fig_name = ['explore the analysis object: ' tree_name];
0166
0167 % Define figure
0168 fig = figure('Units',          'normalized',          ...
0169             'NextPlot',        'add',                ...
0170             'NumberTitle',      'off',                ...
0171             'Toolbar',          'none',               ...
0172             'name',             fig_name,              ...
0173             'Color',            BG_COLOR,              ...
0174             'Position',         [FIG_X FIG_Y          ...
                                FIG_dX FIG_dY]);
0175
0176
0177 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0178 %               Define the tree                       %
0179 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0180
0181 root = ltpda_uitreenode(tree_name, tree_name, [], false);
0182 tree = ltpda_uitree( fig, 'Root',          root,          ...
0183                     'ExpandFcn', @myExpFcn4);
0184 set(tree, 'Units',          'normalized',          ...
0185          'position',        [TREE_X TREE_Y          ...
                                TREE_dX TREE_dY],      ...
0186          'NodeWillExpandCallback', @nodeWillExpand_cb4, ...

```

```

0188         'NodeSelectedCallback', @nodeSelected_cb4);
0189 tmp = tree.FigureComponent;
0190 cell_Data = cell(3,1);
0191
0192 % cell_Data{1} = varargin{:};
0193 cell_Data{1} = obj_value;
0194 cell_Data{3} = obj_name;
0195
0196 set(tmp, 'UserData', cell_Data);
0197
0198
0199 % Define the plot field
0200 haxes = axes('Units', 'normalized', ...
0201             'Position', [PLOT_X PLOT_Y ...
0202                         PLOT_dX PLOT_dY], ...
0203             'Box', 'on', ...
0204             'XTick', [], ...
0205             'YTick', []);
0206
0207 box off;
0208 axis off;
0209
0210 % Define the info fields 'name'
0211 txt1 = uicontrol('String', '', ...
0212                 'Units', 'normalized', ...
0213                 'Style', 'Edit', ...
0214                 'Position', [INFO_X INFO_Y ...
0215                             INFO_dX INFO_dY], ...
0216                 'BackgroundColor', BG_COLOR);
0217 % Define the info fields 'size'
0218 txt2 = uicontrol('String', '', ...
0219                 'Units', 'normalized', ...
0220                 'Style', 'Edit', ...
0221                 'Position', [(INFO_X+INFO_dX) INFO_Y ...
0222                             INFO_dX INFO_dY], ...
0223                 'BackgroundColor', BG_COLOR);
0224 % Define the info fields 'class'
0225 txt3 = uicontrol('String', '', ...
0226                 'Units', 'normalized', ...
0227                 'Style', 'Edit', ...
0228                 'Position', [(INFO_X+2*INFO_dX) INFO_Y ...
0229                             INFO_dX INFO_dY], ...
0230                 'BackgroundColor', BG_COLOR);
0231 % Define the info fields 'value'
0232 txt4 = uicontrol('String', '', ...
0233                 'Units', 'normalized', ...
0234                 'Style', 'Edit', ...
0235                 'Position', [ INFO_X (INFO_Y-INFO_dY) ...
0236                             3*INFO_dX INFO_dY], ...
0237                 'BackgroundColor', BG_COLOR);
0238
0239 % Define the info fields 'value'
0240 txt5 = uicontrol('String', '', ...

```

```

0241         'Units',          'normalized',      ...
0242         'Style',          'listbox',          ...
0243         'Visible',        'off',              ...
0244         'FontSize',        8,                  ...
0245         'Position',        [DISP_X    DISP_Y    ...
0246                             DISP_dX  DISP_dY],    ...
0247         'BackgroundColor', BG_COLOR);
0248
0249
0250
0251 % Define the decription of the info fields
0252 col1 = uicontrol('String',    'Name',          ...
0253                 'Units',      'normalized',      ...
0254                 'Style',      'Text',            ...
0255                 'Position',    [INFO_X    (INFO_Y+INFO_dY)...
0256                                 INFO_dX    INFO_dY],    ...
0257                 'BackgroundColor', BG_COLOR);
0258 col2 = uicontrol('String',    'Size',          ...
0259                 'Units',      'normalized',      ...
0260                 'Style',      'Text',            ...
0261                 'Position',    [(INFO_X+INFO_dX) (INFO_Y+INFO_dY)...
0262                                 INFO_dX          INFO_dY],    ...
0263                 'BackgroundColor', BG_COLOR);
0264 col3 = uicontrol('String',    'Class',          ...
0265                 'Units',      'normalized',      ...
0266                 'Style',      'Text',            ...
0267                 'Position',    [(INFO_X+2*INFO_dX) (INFO_Y+INFO_dY)...
0268                                 INFO_dX          INFO_dY],    ...
0269                 'BackgroundColor', BG_COLOR);
0270
0271 % Define the name of the explorer
0272 expl_name = uicontrol('String',    'Analysis Object explorer', ...
0273                       'Units',      'normalized',      ...
0274                       'Style',      'text',            ...
0275                       'Position',    [EXPL_NAME_X    EXPL_NAME_Y    ...
0276                                       EXPL_NAME_dX    EXPL_NAME_dY], ...
0277                       'ForegroundColor', [0.2 0.4 1],    ...
0278                       'FontSize',      18,                  ...
0279                       'FontWeight',    'bold',              ...
0280                       'FontAngle',     'italic');
0281
0282 tree_menu = uicontextmenu();
0283 tree_menu1 = uimenu(tree_menu, 'Label',    'Plot', ...
0284                           'Callback', @f_tree_menu1);
0285 tree_menu2 = uimenu(tree_menu, 'Label',    'Display', ...
0286                           'Callback', @f_tree_menu2);
0287
0288 disp_menu = uicontextmenu;
0289 disp_menu1 = uimenu(disp_menu, 'Label',    'close', ...
0290                           'Callback', @f_disp_menu1);
0291
0292 set(tree.Tree, 'MousePressedCallback', @mouse_cb);

```

```

0293 set(tree.Tree, 'UIContextMenu', tree_menu);
0294
0295 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0296 % Mouse Pressed Handler
0297     function mouse_cb(h, ev)
0298         if ev.getModifiers()== ev.META_MASK
0299             % Workaround to set the y position
0300             % Workaround to set the x position
0301             vis = get(get(ev, 'Component'), 'VisibleRect');
0302             x_width = get(get(ev, 'Component'), 'Width');
0303             y_height = get(get(ev, 'Component'), 'Height');
0304
0305             new_x = ev.getX-vis(1);
0306             new_y = -ev.getY+y_height-(y_height-vis(4));
0307             set(tree_menu, 'Position', [new_x new_y], ...
0308                 'Visible', 'on');
0309         end
0310
0311     end
0312 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0313     function f_disp_menu1(h, ev)
0314         set(txt5, 'Visible', 'off')
0315         set(tree, 'position', [TREE_X TREE_Y ...
0316             TREE_dX TREE_dY])
0317     end
0318 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0319     function f_tree_menu1(h, ev)
0320         plotselected_cb;
0321     end
0322 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0323     function f_tree_menu2(h, ev)
0324         displayselected_cb;
0325     end
0326 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0327     function plotselected_cb(h, ev)
0328
0329         tmp = tree.FigureComponent;
0330         S = get(tmp, 'UserData');
0331         s = S{1};
0332         cNode = S{2};
0333         [val, plotted, cNode] = getcNodevalue(cNode, s);
0334
0335         cla(haxes)
0336
0337         %%%% Plot history object %%%%
0338         if (isa(val, 'history'))
0339             if length(val) == 1
0340                 figure;
0341                 plot (val);
0342                 ii = strfind(plotted, 'inhists');
0343                 title(haxes, sprintf('History-Level: %d', length(ii)+1))
0344             else

```

```

0345         na = text(0.5,0.5,'Select the left or right branch.');
```

```

0346         set(na, 'HorizontalAlignment', 'center', ...
0347             'Color', 'r', ...
0348             'FontWeight', 'bold', ...
0349             'EdgeColor', 'k', ...
0350             'BackgroundColor', 'w', ...
0351             'FontSize', 10, ...
0352             'Margin', 5);
0353     end
0354
0355     %%%%%%%%% Plot data object %%%%%%%%%
0356     elseif isa(val, 'fsdata') || isa(val, 'tsdata') || ...
0357         isa(val, 'xydata') || isa(val, 'cdata')
0358         figure;
0359         plot(ao(val))
0360
0361     %%%%%%%%% Plot the AO object %%%%%%%%%
0362     elseif isa(val, 'ao')
0363         figure;
0364         plot(ao(val))
0365
0366     %%%%%%%%% Plot mfir and miir object %%%%%%%%%
0367     elseif isa(val, 'mfir') || isa(val, 'miir')
0368         figure;
0369         resp(val)
0370
0371     %%%%%%%%% Plot pzmodel object %%%%%%%%%
0372     elseif isa(val, 'pzmodel')
0373         resp(val)
0374
0375     %%%%%%%%% Is the parent node == 'data' so plot data %%%%%%%%%
0376     else
0377
0378         cNode = S{2};
0379         parent = '';
0380
0381         if cNode.getLevel >= 1
0382             parent = cNode.getParent;
0383         end
0384
0385         if strcmp(parent.getValue, 'data')
0386             grandpa = parent.getParent;
0387             grandpa_val = grandpa.getValue;
0388             von = findstr(grandpa_val, '(');
0389             bis = findstr(grandpa_val, ')');
0390             if ~isempty(von) && ~isempty(bis)
0391                 index = grandpa_val(von+1:bis-1);
0392             else
0393                 index = '1';
0394             end
0395
0396             eval(sprintf('val = S{1}(%s);', index))
0397             figure;

```

```

0398         plot(val);
0399     end
0400
0401 end
0402
0403 end
0404 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0405 function displayselected_cb(h, ev)
0406     tmp = tree.FigureComponent;
0407     S = get(tmp, 'UserData');
0408     s = S{1};
0409     cNode = S{2};
0410     [val, displayed, cNode] = getcNodevalue(cNode, s);
0411
0412     text1 = '';
0413     if isobject(val)
0414         text1 = display(val);
0415     else
0416         disp(val)
0417     end
0418
0419     % some sisplay outputs contains '\n' <-> char(10)
0420     % text can not display this character so replace it with ' '
0421     text1 = strrep(text1, char(10), ' ');
0422
0423     set(tree, 'position', [TREE_X TREE_Y ...
0424         TREE_dX TREE_dY-DISP_dY]);
0425     set(txt5, 'string', text1);
0426     set(txt5, 'Visible', 'on');
0427     set(txt5, 'UIContextMenu', disp_menu);
0428 end
0429 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0430
0431 function cNode = nodeSelected_cb4(tree,ev)
0432     cNode = ev.getCurrentNode;
0433     tmp = tree.FigureComponent;
0434     cell_Data = get(tmp, 'UserData');
0435     cell_Data{2} = cNode;
0436     s = cell_Data{1};
0437     val = s;
0438     plotted = cNode.getValue;
0439     selected = plotted;
0440     [val, plotted, cNode] = getcNodevalue(cNode, val);
0441     set(txt1, 'string', selected)
0442     set(txt2, 'string', strcat(num2str(size(val,1)), 'x', num2str(size
0443 (val,2))) )
0444     set(txt3, 'string', class(val))
0445
0446     str = ' ';
0447     c_str = {};
0448     cla(haxes)

```



```

0449     if ~isempty(val)
0450
0451         if isnumeric(val)
0452             %normalize the vector
0453             si = size(val);
0454             if si(1) > si(2)
0455                 val = val.';
0456             else
0457                 val = val;
0458             end
0459             if size(val,1) == 1 || size(val,2) == 1
0460                 if length(val) > 3
0461                     str = strcat(num2str(val(1:3)), ' ...');
0462                 else
0463                     str = num2str(val);
0464                 end
0465             else
0466                 str = 'Matrix';
0467             end
0468
0469         elseif ischar(val)
0470             str = val;
0471
0472         elseif islogical(val)
0473             if val
0474                 str = 'true';
0475             else
0476                 str = 'false';
0477             end
0478
0479         elseif isobject(val)
0480             if isa(val, 'ao')
0481                 str = 'Analysis Object';
0482             elseif isa(val, 'cdata')
0483                 str = 'C-Data Object';
0484             elseif isa(val, 'fsdata')
0485                 str = 'Frequency-Series Object';
0486             elseif isa(val, 'tsdata')
0487                 str = 'Time-Series Object';
0488             elseif isa(val, 'xydata')
0489                 str = 'X-Y Data Object';
0490
0491             elseif isa(val, 'history')
0492                 str = 'History Object';
0493                 if length(val) == 1
0494                     pl = plist(param('max_depth', MAX_HIST_DEPTH));
0495                     plot(haxes, val, pl);
0496                     ii = strfind(plotted, 'inhists');
0497                     title(haxes, sprintf('History-Level: %d', length(ii)+1))
0498                 else
0499                     c_str{1} = 'Select the left or right branch.';
0500                 end
0501

```

```

0502         elseif isa(val, 'param')
0503             if length(val) == 1
0504                 str = char(val);
0505                 c_str = split_by_comma(str);
0506             else
0507                 for ii = 1:length(val)
0508                     str = char(val(ii));
0509                     c_str1 = split_by_comma(str, '- ');
0510                     c_str(end+1:end+length(c_str1)) = c_str1;
0511                 end
0512             end
0513             str = 'Parameter Object';
0514
0515         elseif isa(val, 'plist')
0516             for ii=1:length(val.params)
0517
0518                 str = char(val.params(ii));
0519                 c_str1 = split_by_comma(str, '- ');
0520                 c_str(end+1:end+length(c_str1)) = c_str1;
0521             end
0522             str = 'Parameter List Object';
0523
0524         elseif isa(val, 'mfir')
0525             str = 'FIR Filter Object';
0526         elseif isa(val, 'miir')
0527             str = 'IIR Filter Object';
0528         elseif isa(val, 'pole')
0529             str = 'Pole Object';
0530         elseif isa(val, 'provenance')
0531             str = 'Provenance Object';
0532         elseif isa(val, 'pzmodel')
0533             str = 'Pole Zero Object';
0534         elseif isa(val, 'specwin')
0535             str = 'Spectral Window Object';
0536         elseif isa(val, 'zero')
0537             str = 'Zero Object';
0538         elseif isa(val, 'time')
0539             str = 'Time Object';
0540         else
0541             str = 'Unknown class';
0542         end
0543
0544     elseif iscell(val)
0545         for i = 1:min(length(val),3)
0546             if ischar(val{i})
0547                 str = strcat(str, val{i});
0548             elseif isnumeric(val)
0549                 str = strcat(str, num2str(val{i}));
0550             end
0551             if i < min(length(val),3)
0552                 str = strcat(str, ',');
0553             end

```

```

0554         end
0555         if length(val) > 3
0556             str = strcat(str, '...');
0557         end
0558
0559     end
0560
0561     if ~isempty(c_str)
0562         c_str = strtrim(c_str);
0563         na = text(0.5,0.5,c_str);
0564         txt_extent = get(na, 'Extent');
0565         set(na, 'Position', [0.5-txt_extent(3)/2, 0.5], ...
0566             'HorizontalAlignment', 'left', ...
0567             'Color', 'k', ...
0568             'FontWeight', 'bold', ...
0569             'EdgeColor', 'k', ...
0570             'BackgroundColor', 'w', ...
0571             'FontSize', 10, ...
0572             'Margin', 5);
0573     end
0574
0575     else % ~isempty(val)
0576         str = 'The field is empty';
0577     end
0578     set(txt4, 'string', str)
0579     set(tmp, 'UserData', cell_Data);
0580 end
0581
0582
0583 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0584 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0585
0586 end
0587
0588 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0589 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0590 function nodes = myExpfcn4(tree,value)
0591
0592     tmp = tree.FigureComponent;
0593     S = get(tmp, 'UserData');
0594     s = S{1};
0595     cNode = S{2};
0596     [val, cNode] = getNodevalue(cNode, s);
0597
0598     % Set the path to the *.gif files
0599     % This tis the current path of this function + 'exp_struct_icons'
0600     pth = '';
0601     eval (sprintf('pth = which('%s');',mfilename))
0602     index = find(pth==filesep, 1, 'last');
0603     pth = pth(1:index);
0604     pth = [pth 'exp_struct_icons' filesep];
0605
0606     [n,m] = size(val);

```

```

0607     count = 0;
0608
0609     %% Vector or Matrix
0610     if m>1 || n>1
0611
0612         if isa(val, 'ao')
0613             iconpath =[pth, 'analysis_object.gif'];
0614         else
0615             iconpath =[pth, 'struct_icon.gif'];
0616         end
0617
0618         %% Vector
0619         if m==1 || n==1
0620             L = length(val);
0621
0622             for J = 1:L
0623                 count = count + 1;
0624                 cNode = S{2};
0625
0626                 level = cNode.getLevel;
0627                 fname = strcat(cNode.getValue, '(' , num2str(J), ')');
0628
0629                 if level==0 && ~isempty(S{3}) && numel(S{3}) == numel(S{1})
0630                     node_str = S{3}(J);
0631                 else
0632                     node_str = fname;
0633                 end
0634
0635                 nodes(count) = ltpda_uitreenode(fname, node_str, iconpath, 0);
0636             end
0637         %% Matrix
0638         else
0639
0640             for ii=1:n
0641                 for jj=1:m
0642                     count = count + 1;
0643                     cNode = S{2};
0644                     fname = [cNode.getValue '(' num2str(ii) ',' num2str(jj) ')'];
0645                     nodes(count) = ltpda_uitreenode(fname, fname, iconpath, 0);
0646                 end
0647             end
0648
0649         end
0650     %% Struct, Object or single value
0651     else
0652         %%
0653         val = val;
0654         fnames = fieldnames(val);
0655
0656         for i=1:length(fnames)
0657             count = count + 1;
0658             x = getfield(val, fnames{i});
0659

```

```

0660         if isa(x, 'ao')
0661             iconpath = [pth, 'analysis_object.gif'];
0662         elseif isa(x, 'tsdata')
0663             iconpath = [pth, 'ts_data.gif'];
0664         elseif isa(x, 'fsdata')
0665             iconpath = [pth, 'fs_data.gif'];
0666         elseif isa(x, 'xydata')
0667             iconpath = [pth, 'xy_data.gif'];
0668         elseif isa(x, 'cdata')
0669             iconpath = [pth, 'c_data.gif'];
0670         elseif isa(x, 'history')
0671             iconpath = [pth, 'history.gif'];
0672         elseif isa(x, 'plist')
0673             iconpath = [pth, 'plist.gif'];
0674
0675         elseif isstruct(x)
0676             if length(x) > 1
0677                 iconpath = [pth, 'structarray_icon.gif'];
0678             else
0679                 iconpath = [pth, 'struct_icon.gif'];
0680             end
0681         elseif isnumeric(x)
0682             iconpath = [pth, 'double_icon.gif'];
0683         elseif iscell(x)
0684             iconpath = [pth, 'cell_icon.gif'];
0685         elseif ischar(x)
0686             iconpath = [pth, 'char_icon.gif'];
0687         elseif islogical(x)
0688             iconpath = [pth, 'logic_icon.gif'];
0689         elseif isobject(x)
0690             iconpath = [pth, 'obj_icon.gif'];
0691         else
0692             iconpath = [pth, 'unknown_icon.gif'];
0693         end
0694
0695         if isstruct(x) || isobject(x)
0696             isLeaf = 0;
0697         else
0698             isLeaf = 1;
0699         end
0700
0701         nodes(count) = ltpda_uitreenode(fnames{i}, fnames{i}, iconpath,
isLeaf);
0702     end
0703 end
0704
0705     if (count == 0)
0706         nodes = [];
0707     end
0708 end
0709
0710 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0711

```

```

0712     function cNode = nodeWillExpand_cb4(tree,ev)
0713         cNode = ev.getCurrentNode;
0714         tmp = tree.FigureComponent;
0715         cell_Data = get(tmp, 'UserData');
0716         cell_Data{2} = cNode;
0717         set(tmp, 'UserData', cell_Data);
0718     end
0719
0720 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0721     function [val, displayed, cNode] = getNodevalue(cNode, s)
0722
0723         fields = {};
0724         while cNode.getLevel ~=0
0725             fields = [fields; cNode.getValue];
0726             c = findstr(cNode.getValue, '(');
0727             if ~isempty(c) && cNode.getLevel ~=0
0728                 cNode = cNode.getParent;
0729             end
0730
0731             if cNode.getLevel ==0, break; end
0732             cNode = cNode.getParent;
0733         end
0734
0735         val = s;
0736
0737         if ~isempty(fields)
0738             L=length(fields);
0739             displayed = fields{L};
0740             % create the variable: displayed
0741             for j = L-1:-1:1
0742                 displayed = strcat(displayed, '.', fields{j});
0743             end
0744
0745             for i = L:-1:1
0746                 field = fields{i};
0747                 von = findstr(field, '(');
0748                 bis = findstr(field, ')');
0749                 if ~isempty(von)
0750
0751                     idx = field(von+1:bis-1);
0752                     field = field(1:von-1);
0753                     if (strcmp(field, cNode.getValue))
0754                         cmd = sprintf('val = val(%s);',idx);
0755                         eval(cmd);
0756                         if iscell(val) && numel(val) == 1
0757                             val = val{1};
0758                         else
0759                             error('##### MAch mich neu');
0760                         end
0761                     else
0762
0763                         cmd = sprintf('val = getfield(val, field, {%s});',idx);

```

```

0764         eval(cmd);
0765         if iscell(val) && numel(val) == 1
0766             val = val{1};
0767         end
0768
0769     end
0770
0771     else
0772         if iscell(val) && numel(val) == 1
0773             val = val{1};
0774         elseif numel(val) ~= 1
0775             error('##### MAch mich neu');
0776         end
0777         val = getfield(val, field);
0778     end
0779 end
0780 else
0781     displayed = cNode.getValue;
0782     if iscell(val) && numel(val) == 1
0783         val = val{1};
0784     else
0785         end
0786     return;
0787 end
0788 end
0789
0790
0791 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0792 function c_str = split_by_comma(str, pref)
0793
0794     if nargin < 2
0795         pref = '';
0796     end
0797
0798     c_str = {};
0799     c_str{1} = str;
0800     index    = find(str==' , ');
0801     von      = 1;
0802     for ii = 1:length(index)
0803         bis = index(ii)-1;
0804         c_str{ii} = [pref str(von:bis)];
0805         von = bis + 2;
0806     end
0807     if ~isempty(index)
0808         c_str{ii+1} = [pref str(von:end)];
0809     end
0810
0811     c_str = strtrim(c_str);
0812 end
0813

```


[Home](#) > [m](#) > [sigproc](#) > [time_domain](#) > ltpda_timedomainfit.m

ltpda_timedomainfit

PURPOSE

LTPDA_TIMEDOMAINFIT uses `lsconv` to fit a set of time-series AOs to a target time-series AO.

SYNOPSIS

function varargout = ltpda_timedomainfit(varargin)

DESCRIPTION

LTPDA_TIMEDOMAINFIT uses `lsconv` to fit a set of time-series AOs to a target time-series AO.

%%%

DESCRIPTION: LTPDA_TIMEDOMAINFIT uses `lsconv` 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

```

0001 function varargout = ltpda_timedomainfit(varargin)
0002 % LTPDA_TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to a target
time-series AO.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to
0007 %               a target time-series AO.
0008 %
0009 % CALL:        b = ltpda_timedomainfit(a1,a2,a3,..., pl)
0010 %
0011 % INPUTS:      asN    - a list of analysis objects containing the data to be
fitted.
0012 %               The first input analysis object that contains time-series
0013 %               data is the target object. A linear combination of all
other
0014 %               time-series AOs is fit to this target.
0015 %               pl     - a parameter list
0016 %
0017 %
0018 % PARAMETERS:  no valid parameters at the moment.
0019 %
0020 % OUTPUTS:     b - cdata type analysis object containing the fitted coefficients
0021 %
0022 % GUI CALL:    The following call returns a parameter list object that contains
0023 %               the default parameter values:
0024 %
0025 %               >> pl = ltpda_xcorr('Params')
0026 %
0027 % VERSION:     $Id: ltpda_timedomainfit.m,v 1.5 2007/11/08 10:35:04 hewitson
Exp $
0028 %
0029 % HISTORY:     01-03-2007 M Hewitson
0030 %               Creation
0031 %
0032 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0033 %
0034 % Check if this is a call for parameters
0035 if nargin == 1
0036     in = char(varargin{1});

```

```

0037     if strcmp(in, 'Params')
0038         varargout{1} = getDefaultPL();
0039         return
0040     end
0041 end
0042
0043 % capture input variable names
0044 invars = {};
0045 for j=1:nargin
0046     iname = inputname(j);
0047     if isempty(iname) & isnumeric(varargin{j})
0048         iname = num2str(varargin{j});
0049     elseif isempty(iname) & ischar(varargin{j})
0050         iname = varargin{j};
0051     end
0052     invars = [invars cellstr(iname)];
0053 end
0054
0055 ALGONAME = mfilename;
0056 VERSION  = '$Id: ltpda_timedomainfit.m,v 1.5 2007/11/08 10:35:04 hewitson Exp $';
0057
0058 % Look at inputs
0059 if nargin < 1
0060     error('### Incorrect number of inputs.')
0061 end
0062
0063 as = [];
0064 pl = [];
0065 for j=1:nargin
0066     a = varargin{j};
0067     if isa(a, 'plist')
0068         pl = a;
0069     end
0070     if isa(a, 'ao')
0071         for k=1:length(a)
0072             ak = a(k);
0073             d = ak.data;
0074             if isa(d, 'tsdata')
0075                 as = [as ak];
0076             else
0077                 warning('### skipping fsdata/A0.');

```

```

0089 naFit = length(as);
0090 if naFit < 2
0091     error('### Please input at least two A0s.');
```

0092 end

0093

```

0094 ho = [];
0095 for i=1:naFit
0096
0097     % get data out
0098     a = as(i);
0099     ho = [ho a.hist];
0100     d = a.data;
0101     dinfo = whos('d');
```

0102 if ~isreal(d.y)

```

0103     error('### I only work with real time-series at the moment.');
```

0104 end

```

0105 x(:,i) = d.y;
0106 end
0107 % Fit
0108 [m,n] = size(x);
0109 [coef, stdx, mse, S] = lscov(x(:,2:n),x(:,1));
0110
0111 disp(['*** got coefficients: ' num2str(coef.)]);
0112
0113 % Return new cdata A0 with the coefficients in
0114 % - we add '1' to the front of the coefficients so that ltpda_lincom
0115 % will work directly.
0116
0117 % make a new history object
0118 h = history(ALGONAME, VERSION, [], ho);
0119 h = set(h, 'invars', invars);
0120
0121 % make output analysis object
0122 bo = ao([1 -coef.]);
0123 bo = setnh(bo, 'name', ['fit of [' sprintf('%s ', char(invars{2:end})) ']' to '
char(invars{1})]);
0124 bo = setnh(bo, 'hist', h);
0125
0126 varargout{1} = bo;
0127
0128 %-----
0129 % Get default params
0130 function plo = getDefaultPL()
0131
0132 disp('* creating default plist...');
0133 plo = plist();
0134 disp('* done.');
```

0135

0136

```

0137 % END
```


[Home](#) > [classes](#) > [@tsdata](#) > setTime.m

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 1.6 2007/11/30 19:36:05 ingo 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
0003 % vector and the sample rate.
```

```

0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SETTIME sets the time-vector of a tsdata object based on the
0007 %                length of the x vector and the sample rate.
0008 %
0009 % CALL:         tsdata = setTime(tsdata)
0010 %
0011 % VERSION:      $Id: setTime.m,v 1.6 2007/11/30 19:36:05 ingo Exp $
0012 %
0013 % HISTORY:      31-01-2007 M Hewitson
0014 %                Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: setTime.m,v 1.6 2007/11/30 19:36:05 ingo Exp $';
0019
0020 if nargin == 2 && isa(ts, 'tsdata') && ischar(varargin{1})
0021     if strcmp(varargin{1}, 'Params')
0022         ts = plist();
0023         return
0024     elseif strcmp(varargin{1}, 'Version')
0025         ts = VERSION;
0026         return
0027     end
0028 end
0029
0030 ts.x      = linspace(0, ts.nsecs-1/ts.fs, length(ts.y));
0031 ts        = reshapeT(ts);
0032
0033 % END

```

[Home](#) > [classes](#) > [@tsdata](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a tsdata from a tsdata xml file.

SYNOPSIS

function ps = xmlparse(p, xmlfile)

DESCRIPTION

XMLPARSE creates a tsdata from a tsdata xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a param object into a string.

This function is called by:

- [tsdata](#) TSDATA time-series object class constructor.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a tsdata from a tsdata xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```



```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'tsdata')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```

[Home](#) > [m](#) > [gui](#) > [ao_browser](#) > ltpda_uitree.m

ltpda_uitree

PURPOSE

WARNING: This feature is not supported in MATLAB

SYNOPSIS

function [tree, container] = ltpda_uitree(varargin)

DESCRIPTION

WARNING: This feature is not supported in MATLAB and the API and functionality may change in a future release.

UITREE creates a uitree component with hierarchical data in a figure window. UITREE creates an empty uitree object with default property values in a figure window.

UITREE('PropertyName1', 'Value1', 'PropertyName2', 'Value2', ...) creates a uitree object with the specified properties. The properties that can be set are: Root, ExpandFcn, SelectionChangeFcn, Parent and Position. The 'Root' property must be specified to successfully to create a uitree. The other properties are optional.

UITREE(figurehandle, ...) creates a uitree object in the figure window specified by the figurehandle.

HANDLE = UITREE(...) creates a uitree object and returns its handle.

Properties:

Root - Root node for the uitree object. Could be handle to a HG object, a string, an open block diagram name, or handle to a UITREENODE object.

ExpandFcn - Node expansion function. String or function handle.

SelectionChangeFcn - Selection callback function. String or function handle.

Parent - Parent figure handle. If not specified, it is the gcf.

Position: 4 element vector specifying the position.

DndEnabled: Boolean specifying if drag and drop is enabled (false).

MultipleSelectionEnabled: Boolean specifying if multiple selection is allowed (false).

SelectedNodes: vector of uitreenodes to be selected.

Units: String - pixels/normalized/inches/points/centimeters.

Visible: Boolean specifying if table is visible.

NodeDroppedCallback: Callback for a drag and drop action.
 NodeExpandedCallback: Callback for a node expand action.
 NodeCollapsedCallback: Callback function for a node collapse action.
 NodeSelectedCallback: Callback for a node selection action.

Examples:

```
t = uitree('Root', 'D:\')
```

%Creates a uitree widget in a figure window with which acts as a
 %directory browser with the D: drive as the root node.

```
surf(peaks)
f = figure
t = uitree(f, 'Root', 0)
```

%Creates a uitree object in the specified figure window which acts as
 %a MATLAB hierarchy browser with the MATLAB root (0) as the root node.

```
root = uitreenode('S:\', 'S', [], false);
t = uitree('Root', root, 'ExpandFcn', @myExpFcn, ...
    'SelectionChangeFcn', 'disp(''Selection Changed'')');
```

%Creates a uitree object with the specified root node and a custom
 %function to return child nodes for any given node. The function
 %myExpFcn is an m-file in the MATLAB path with the following code:

%This function should be added to your path

% -----

```
function nodes = myExpFcn(tree, value)
```

```
try
```

```
    count = 0;
```

```
    ch = dir(value);
```

```
    for i=1:length(ch)
```

```
        if (any(strcmp(ch(i).name, {'.', '..', ''})) == 0)
```

```
            count = count + 1;
```

```
            if ch(i).isdir
```

```
                iconpath = [matlabroot, '/toolbox/matlab/icons/foldericon.
```

```
gif'];
```

```
            else
```

```
                iconpath = [matlabroot, '/toolbox/matlab/icons/pageicon.gif'];
```

```
            end
```

```
            nodes(count) = uitreenode([value, ch(i).name, filesep], ...
```

```
                ch(i).name, iconpath, ~ch(i).isdir);
```

```
        end
```

```
    end
```

```
catch
```

```
    error(['The uitree node type is not recognized. You may need to ', ...
        'define an ExpandFcn for the nodes.']);
```

```
end
```

```

    if (count == 0)
        nodes = [];
    end
% -----

```

See also UITREENODE, UITABLE, PATH

Copyright 2003-2006 The MathWorks, Inc.
\$Revision: 1.1 \$ \$Date: 2007/06/14 14:53:32 \$

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

```

0001 function [tree, container] = ltpda_uitree(varargin)
0002 % WARNING: This feature is not supported in MATLAB
0003 % and the API and functionality may change in a future release.
0004 %
0005 % UITREE creates a uitree component with hierarchical data in a figure window.
0006 %   UITREE creates an empty uitree object with default property values in
0007 %   a figure window.
0008 %
0009 %   UITREE('PropertyName1', 'Value1', 'PropertyName2', 'Value2', ...)
0010 %   creates a uitree object with the specified properties. The properties
0011 %   that can be set are: Root, ExpandFcn, SelectionChangeFcn, Parent and
0012 %   Position. The 'Root' property must be specified to successfully to
0013 %   create a uitree. The other properties are optional.
0014 %
0015 %   UITREE(figurehandle, ...) creates a uitree object in the figure
0016 %   window specified by the figurehandle.
0017 %
0018 %   HANDLE = UITREE(...) creates a uitree object and returns its handle.
0019 %

```

```

0020 %   Properties:
0021 %
0022 %   Root - Root node for the uitree object. Could be handle to a HG
0023 %   object, a string, an open block diagram name, or handle to a
0024 %   UITREENODE object.
0025 %   ExpandFcn - Node expansion function. String or function handle.
0026 %   SelectionChangeFcn - Selection callback function. String or function
0027 %   handle.
0028 %   Parent - Parent figure handle. If not specified, it is the(gcf.
0029 %   Position: 4 element vector specifying the position.
0030 %
0031 %   DndEnabled: Boolean specifying if drag and drop is enabled (false).
0032 %   MultipleSelectionEnabled: Boolean specifying if multiple selection is
0033 %   allowed (false).
0034 %   SelectedNodes: vector of uitreenodes to be selected.
0035 %   Units: String - pixels/normalized/inches/points/centimeters.
0036 %   Visible: Boolean specifying if table is visible.
0037 %   NodeDroppedCallback: Callback for a drag and drop action.
0038 %   NodeExpandedCallback: Callback for a node expand action.
0039 %   NodeCollapsedCallback: Callback function for a node collapse action.
0040 %   NodeSelectedCallback: Callback for a node selection action.
0041 %
0042 %
0043 %   Examples:
0044 %       t = uitree('Root', 'D:\')
0045 %
0046 %       %Creates a uitree widget in a figure window with which acts as a
0047 %       %directory browser with the D: drive as the root node.
0048 %
0049 %       surf(peaks)
0050 %       f = figure
0051 %       t = uitree(f, 'Root', 0)
0052 %
0053 %       %Creates a uitree object in the specified figure window which acts as
0054 %       %a MATLAB hierarchy browser with the MATLAB root (0) as the root node.
0055 %
0056 %       root = uitreenode('S:\', 'S', [], false);
0057 %       t = uitree('Root', root, 'ExpandFcn', @myExpFcn, ...
0058 %       'SelectionChangeFcn', 'disp(''Selection Changed'')');
0059 %
0060 %       %Creates a uitree object with the specified root node and a custom
0061 %       %function to return child nodes for any given node. The function
0062 %       %myExpFcn is an m-file in the MATLAB path with the following code:
0063 %
0064 %       %This function should be added to your path
0065 %       % -----
0066 %       function nodes = myExpFcn(tree, value)
0067 %
0068 %       try
0069 %           count = 0;
0070 %           ch = dir(value);
0071 %
0072 %           for i=1:length(ch)

```

```

0073 %             if (any(strcmp(ch(i).name, {'.', '..', ''})) == 0)
0074 %                 count = count + 1;
0075 %                 if ch(i).isdir
0076 %                     iconpath = [matlabroot, '/toolbox/matlab/icons/
foldericon.gif'];
0077 %                 else
0078 %                     iconpath = [matlabroot, '/toolbox/matlab/icons/pageicon.
gif'];
0079 %                 end
0080 %                 nodes(count) = uitreenode([value, ch(i).name, filesep], ...
0081 %                     ch(i).name, iconpath, ~ch(i).isdir);
0082 %             end
0083 %         end
0084 %     catch
0085 %         error(['The uitree node type is not recognized. You may need to
', ...
0086 %             'define an ExpandFcn for the nodes.']);
0087 %     end
0088 %
0089 %     if (count == 0)
0090 %         nodes = [];
0091 %     end
0092 %     % -----
0093 %
0094 %     See also UITREENODE, UITABLE, PATH
0095 %
0096 %     Copyright 2003-2006 The MathWorks, Inc.
0097 %     $Revision: 1.1 $   $Date: 2007/06/14 14:53:32 $
0098
0099 %     Release: R14. This feature will not work in previous versions of MATLAB.
0100
0101 %% Setup and P-V parsing.
0102
0103 error(javachk('awt'));
0104 error(nargoutchk(0, 2, nargout));
0105
0106 fig = [];
0107 numargs = nargin;
0108
0109 if (nargin > 0 && isscalar(varargin{1}) && ishandle(varargin{1}))
0110     if ~isa(handle(varargin{1}), 'figure')
0111         error('MATLAB:uitree:InvalidFigureHandle', 'Unrecognized parameter.');
```

```

0123 % parent = [];
0124
0125 if (numargs == 1)
0126     error('MATLAB:uitree:InvalidNumInputs', 'Unrecognized parameter.');
```

0127 end

0128

```

0129 for i = 1:2:numargs-1
0130     if ~ischar(varargin{i})
0131         error('MATLAB:uitree:UnrecognizedParameter', 'Unrecognized parameter.');
```

0132

```

0133     end
0134     switch lower(varargin{i})
0135         case 'root'
0136             root = varargin{i+1};
0137         case 'expandfcn'
0138             expfcn = varargin{i+1};
0139         case 'selectionchangeFcn'
0140             selfcn = varargin{i+1};
0141         case 'parent'
0142             if ishandle(varargin{i+1})
0143                 f = varargin{i+1};
0144                 if isa(handle(f), 'figure')
0145                     fig = f;
0146                 end
0147             end
0148         case 'position'
0149             p = varargin{i+1};
0150             if isnumeric(p) && (length(p) == 4)
0151                 pos = p;
0152             end
0153         otherwise
0154             error('MATLAB:uitree:UnknownParameter', ['Unrecognized parameter:
', varargin{i}]);
0155     end
0156 end
0157
0158 if isempty(expfcn)
0159     [root, expfcn] = processNode(root);
0160 else
0161     root = processNode(root);
0162 end
0163
0164 tree_h = com.mathworks.hg.peer.UITreePeer;
0165 tree_h.setRoot(root);
0166
0167 if isempty(fig)
0168     fig = gcf;
0169 end
0170
0171 if isempty(pos)
0172     figpos = get(fig, 'Position');
0173     pos = [0 0 min(figpos(3), 200) figpos(4)];
0174 end
```

```

0175
0176 % pass the figure child in, let javacomponent introspect
0177 [obj, container] = javacomponent(tree_h, pos, fig);
0178 % javacomponent returns a UDD handle for the java component passed in.
0179 tree = obj;
0180
0181 if ~isempty(expfcn)
0182     set(tree, 'NodeExpandedCallback', {@nodeExpanded, tree, expfcn});
0183 end
0184
0185 if ~isempty(selfcn)
0186     set(tree, 'NodeSelectedCallback', {@nodeSelected, tree, selfcn});
0187 end
0188
0189 end
0190
0191 %% -----
0192 function nodeExpanded(src, evd, tree, expfcn)                                %#ok
0193
0194 % tree = handle(src);
0195 % evdsrc = evd.getSource;
0196
0197 evdnode = evd.getCurrentNode;
0198 % indices = [];
0199
0200 if ~tree.isLoaded(evdnode)
0201     value = evdnode.getValue;
0202
0203     % <call a user function(value) which returns uitreenodes>;
0204     cbk = expfcn;
0205     if iscell(cbk)
0206         childnodes = feval(cbk{1}, tree, value, cbk{2:end});
0207     else
0208         childnodes = feval(cbk, tree, value);
0209     end
0210
0211     if (length(childnodes) == 1)
0212         % Then we dont have an array of nodes. Create an array.
0213         chnodes = childnodes;
0214         childnodes = javaArray('com.mathworks.hg.peer.UITreeNode', 1);
0215         childnodes(1) = java(chnodes);
0216     end
0217
0218     tree.add(evdnode, childnodes);
0219     tree.setLoaded(evdnode, true);
0220 end
0221
0222 end
0223
0224 %% -----
0225 function nodeSelected(src, evd, tree, selfcn)                                %#ok
0226 cbk = selfcn;

```



```

0227 hgfeval(cbk, tree, evd);
0228
0229 end
0230
0231 %% -----
0232 function [node, expfcn] = processNode(root)
0233 expfcn = [];
0234
0235 if isempty(root) || isa(root, 'com.mathworks.hg.peer.UITreeNode') || ...
0236     isa(root, 'javahandle.com.mathworks.hg.peer.UITreeNode')
0237     node = root;
0238 elseif ishghandle(root)
0239     % Try to process as an HG object.
0240     try
0241         node = uitreenode(handle(root), get(root, 'Type'), ...
0242             [], isempty(get(0, 'children')));
0243     catch
0244         node = [];
0245     end
0246     expfcn = @hgBrowser;
0247 elseif ismodel(root)
0248     % Try to process as an open Simulink system
0249
0250     % TODO if there is an open simulink system and a directory on the path with
0251     % the same name, the system will hide the directory. Perhaps we should
0252     % warn about this.
0253     try
0254         h = handle(get_param(root, 'Handle'));
0255         % TODO we pass root to the tree as a string,
0256         % it would be better if we could just pass the
0257         % handle up
0258         node = uitreenode(root, get(h, 'Name'), ...
0259             [], isempty(h.getHierarchicalChildren));
0260     catch
0261         node = [];
0262     end
0263     expfcn = @mdlBrowser;
0264 elseif ischar(root)
0265     % Try to process this as a directory structure.
0266     try
0267         iconpath = [matlabroot, '/toolbox/matlab/icons/foldericon.gif'];
0268         node = uitreenode(root, root, iconpath, ~isdir(root));
0269     catch
0270         node = [];
0271     end
0272     expfcn = @dirBrowser;
0273 else
0274     node = [];
0275 end
0276
0277 end
0278

```

```

0279 %% -----
0280 function nodes = hgBrowser(tree, value)                                %#ok
0281
0282 try
0283     count = 0;
0284     parent = handle(value);
0285     ch = parent.children;
0286
0287     for i=1:length(ch)
0288         count = count+1;
0289         nodes(count) = uitreenode(handle(ch(i)), get(ch(i), 'Type'), [], ...
0290             isempty(get(ch(i), 'children')));
0291     end
0292 catch
0293     error('MATLAB:uitree:UnknownNodeType', ['The uitree node type is not
0294 recognized. You may need to ', ...
0295     'define an ExpandFcn for the nodes.']);
0296 end
0297 if (count == 0)
0298     nodes = [];
0299 end
0300
0301 end
0302
0303 %% -----
0304 function nodes = mdlBrowser(tree, value)                                %#ok
0305
0306 try
0307     count = 0;
0308     parent = handle(get_param(value, 'Handle'));
0309     ch = parent.getHierarchicalChildren;
0310
0311     for i=1:length(ch)
0312         if isempty(findstr(class(ch(i)), 'SubSystem'))
0313             % not a subsystem
0314         else
0315             % is a subsystem
0316             count = count+1;
0317             descr = get(ch(i), 'Name');
0318             isleaf = true;
0319             cch = ch(i).getHierarchicalChildren;
0320             if ~isempty(cch)
0321                 for j = 1:length(cch)
0322                     if ~isempty(findstr(class(cch(j)), 'SubSystem'))
0323                         isleaf = false;
0324                         break;
0325                     end
0326                 end
0327             end
0328             nodes(count) = uitreenode([value '/' descr], descr, [], ...
0329                 isleaf);
0330         end
0331     end

```

```

0331     end
0332 catch
0333     error('MATLAB:uitree:UnknownNodeType', ['The uitree node type is not
recognized. You may need to ', ...
0334         'define an ExpandFcn for the nodes.']);
0335 end
0336
0337 if (count == 0)
0338     nodes = [];
0339 end
0340
0341 end
0342
0343
0344 %% -----
0345 function nodes = dirBrowser(tree, value)                                %#ok
0346
0347 try
0348     count = 0;
0349     ch = dir(value);
0350
0351     for i=1:length(ch)
0352         if (any(strcmp(ch(i).name, {'.', '..', '}')) == 0)
0353             count = count + 1;
0354             if ch(i).isdir
0355                 iconpath = [matlabroot, '/toolbox/matlab/icons/foldericon.gif'];
0356             else
0357                 iconpath = [matlabroot, '/toolbox/matlab/icons/pageicon.gif'];
0358             end
0359             nodes(count) = uitreenode([value, ch(i).name, filesep], ...
0360                 ch(i).name, iconpath, ~ch(i).isdir);
0361         end
0362     end
0363 catch
0364     error('MATLAB:uitree:UnknownNodeType', ['The uitree node type is not
recognized. You may need to ', ...
0365         'define an ExpandFcn for the nodes.']);
0366 end
0367
0368 if (count == 0)
0369     nodes = [];
0370 end
0371
0372 end
0373
0374 %% -----
0375 function yesno = ismodel(input)
0376 yesno = false;
0377
0378 try
0379     get_param(input, 'handle');
0380     yesno = true;
0381 catch

```

0382 [end](#)

0383

0384 [end](#)

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > setWorkspaceObjsList.m

setWorkspaceObjsList

PURPOSE

SYNOPSIS

```
function setWorkspaceObjsList(objs)
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [cb_refreshObjsList](#) get a list of objects

SOURCE CODE

```
0001 function setWorkspaceObjsList(objs)
0002
0003 id = findobj('Tag', 'workspaceObjsList');
0004
0005 objlist = [];
0006 for j=1:length(objs)
0007     obj = objs(j);
0008     str = sprintf('%s\t\t(%s)', obj.name, obj.class);
0009     objlist = [objlist cellstr(str)];
0010 end
0011
0012 set(id, 'Value', 1);
0013 set(id, 'String', objlist);
0014
```

0015

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@xydata](#) > xmlparse.m

xmlparse

PURPOSE

XMLPARSE creates a xydata from a xydata xml file.

SYNOPSIS

function ps = xmlparse(p, xmlfile)

DESCRIPTION

XMLPARSE creates a xydata from a xydata xml file.

M Hewitson 28-08-07

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a param object into a string.

This function is called by:

- [xydata](#) XYDATA X-Y data object class constructor.

SOURCE CODE

```
0001 function ps = xmlparse(p, xmlfile)
0002
0003 % XMLPARSE creates a xydata from a xydata xml file.
0004 %
0005 % M Hewitson 28-08-07
0006 %
0007
```

```
0008 % Read in XML
0009 xdoc = xmlread(xmlfile);
0010
0011 objs = xdoc.getElementsByTagName('Object');
0012 obj = objs.item(0);
0013
0014 children = obj.getChildNodes;
0015 ps = [];
0016 % create param from each child
0017 for j=1:children.getLength
0018     child = children.item(j-1);
0019     if strcmp(char(child.getNodeName), 'xydata')
0020         ps = [ps fromxml(child)];
0021     end
0022 end
0023
0024 end
0025
0026 % END
```


[Home](#) > [m](#) > [gui](#) > [ao_browser](#) > ltpda_uitreenode.m

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.

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

```
0001 function node = ltpda_uitreenode(value, string, icon, isLeaf)
0002 % WARNING: This feature is not supported in MATLAB
0003 % and the API and functionality may change in a future release.
0004 %
0005 %     UITREENODE(Value, Description, Icon, Leaf)
0006 %     creates a tree node object for the uitree with the specified
0007 %     properties. All properties must be specified for the successful
```

```

0008 %   creation of a node object.
0009 %
0010 %   Value can be a string or handle represented by this node.
0011 %   Description is a string which is used to identify the node.
0012 %   Icon can be a qualified pathname to an image to be used as an icon
0013 %   for this node. It may be set to [] to use default icons.
0014 %   Leaf can be true or false to denote whether this node has children.
0015 %
0016 %   Example:
0017 %       t = uitree('Root', 'D:\')
0018 %
0019 %       %Creates a uitree widget in a figure window with which acts as a
0020 %       %directory browser with the D: drive as the root node.
0021 %
0022 %       surf(peaks)
0023 %       f = figure
0024 %       t = uitree(f, 'Root', 0)
0025 %
0026 %       %Creates a uitree object in the specified figure window which acts as
0027 %       %a MATLAB hierarchy browser with the MATLAB root (0) as the root node.
0028 %
0029 %       root = uitreenode('S:\', 'S', [], false);
0030 %       t = uitree('Root', root, 'ExpandFcn', @myExpfcn, ...
0031 %               'SelectionChangeFcn', 'disp(''Selection Changed'')');
0032 %
0033 %       %Creates a uitree object with the specified root node and a custom
0034 %       %function to return child nodes for any given node. The function
0035 %       %myExpfcn is a user defined m-file in the MATLAB path.
0036 %
0037 %       % This function should be added to the path
0038 %       % -----
0039 %       function nodes = myExpfcn(tree, value)
0040 %
0041 %       try
0042 %           count = 0;
0043 %           ch = dir(value);
0044 %
0045 %           for i=1:length(ch)
0046 %               if (any(strcmp(ch(i).name, {'.', '..', ''})) == 0)
0047 %                   count = count + 1;
0048 %                   if ch(i).isdir
0049 %                       iconpath = [matlabroot, '/toolbox/matlab/icons/foldericon.
0050 %                       gif'];
0051 %                   else
0052 %                       iconpath = [matlabroot, '/toolbox/matlab/icons/pageicon.
0053 %                       gif'];
0054 %                   end
0055 %                   nodes(count) = uitreenode([value, ch(i).name, filesep], ...
0056 %                           ch(i).name, iconpath, ~ch(i).isdir);
0057 %               end
0058 %           end
0059 %       catch
0060 %           error(['The uitree node type is not recognized. You may need to ', ...

```

```
0059 %             'define an ExpandFcn for the nodes.']);
0060 %         end
0061 %
0062 %         if (count == 0)
0063 %             nodes = [];
0064 %         end
0065 %         % -----
0066 %
0067 %     See also UITREE, UITABLE, JAVACOMPONENT
0068 %
0069 % Copyright 2003-2006 The MathWorks, Inc.
0070
0071 import com.mathworks.hg.peer.UITreeNode;
0072 node = handle(UITreeNode(value, string, icon, isLeaf));
0073 schema.prop(node, 'UserData', 'MATLAB array');
```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > setWorkspaceObjsListQL.m

setWorkspaceObjsListQL

PURPOSE

SYNOPSIS

```
function setWorkspaceObjsList(objs)
```

DESCRIPTION

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [cb_refreshObjsListQL](#) get a list of objects

SOURCE CODE

```
0001 function setWorkspaceObjsList(objs)
0002
0003 id = findobj('Tag', 'workspaceObjsListQL');
0004
0005 objlist = [];
0006 for j=1:length(objs)
0007     obj = objs(j);
0008     str = sprintf('%s\t\t(%s)', obj.name, obj.class);
0009     objlist = [objlist cellstr(str)];
0010 end
0011
0012 set(id, 'Value', 1);
0013 set(id, 'String', objlist);
0014
```

0015

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > fillClassMethodsList.m

fillClassMethodsList

PURPOSE

Get the object types

SYNOPSIS

```
function fillClassMethodsList(h)
```

DESCRIPTION

Get the object types

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- [cb_workspaceObjsListQL](#)

SUBFUNCTIONS

- [function type = getClassFromString\(str\)](#)

SOURCE CODE

```
0001 function fillClassMethodsList(h)
0002
0003 % Get the object types
0004 id = findobj('Tag', 'workspaceObjsListQL');
0005 ostr = get(id, 'String');
0006 oidx = get(id, 'Value');
```

```
0007
0008 objStrs = ostr(oidx);
0009
0010 if ~isempty(objStrs)
0011
0012     type = getClassFromString(objStrs{1});
0013     diff = 0;
0014     for j=2:length(objStrs)
0015         if ~strcmp(type, getClassFromString(objStrs{j}))
0016             diff = 1;
0017         end
0018     end
0019
0020     mths = {};
0021     if ~diff
0022         tmps = methods(type);
0023         % Remove exceptions
0024         for j=1:length(tmps)
0025             if ~strcmp(tmps{j}, 'Contents')
0026                 % &&...
0027                 % ~strcmp(tmps{j}, 'subsref')
0028                 mths = [mths tmps(j)];
0029             end
0030         end
0031     end
0032
0033     % Fill list
0034     id = findobj('Tag', 'classMethodList');
0035     set(id, 'Value', 1);
0036     set(id, 'String', mths);
0037
0038 end
0039
0040 function type = getClassFromString(str)
0041
0042 [s,r] = strtok(str, '(');
0043 type = r(2:end-1);
```


[Home](#) > [m](#) > [helper](#) > ltpda_unwrapdeg.m

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 %
```

```
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_UNWRAPDEG Unwrap a phase vector given in degrees.
0007 %
0008 % CALL:          deg = ltpda_unwrapdeg(phase)
0009 %
0010 % INPUTS:        phase - phase vector
0011 %
0012 % OUTPUTS:       deg   - degrees vector
0013 %
0014 % VERSION: $Id: ltpda_unwrapdeg.m,v 1.5 2007/07/13 12:17:39 ingo Exp $
0015 %
0016 % HISTORY: 26-01-2007 M Hewitson
0017 %           Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 deg = unwrap(phase*pi/180)*180/pi;
0022
0023 % END
```

[Home](#) > [classes](#) > [@cdata](#) > set_xy_axis.m

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 1.3 2007/11/28 13:54:00 ingo 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

```
0001 function data_out = set_xy_axis(data, x, y)
0002 % SET_XY_AXIS set the x-axis and y-axis values.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET_XY_AXIS set the x-axis and y-axis values.
```

```

0007 %
0008 % CALL:      data = set_xy_axis (data,    x, y);
0009 %          ao.data = set_xy_axis (ao.data, x, y);
0010 %
0011 % VERSION: $Id: set_xy_axis.m,v 1.3 2007/11/28 13:54:00 ingo Exp $
0012 %
0013 % HISTORY: 21-06-2007 Diepholz
0014 %          Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: set_xy_axis.m,v 1.3 2007/11/28 13:54:00 ingo Exp $';
0019
0020 % Check if this is a call for parameters
0021 if nargin == 2
0022     if isa(data, 'cdata') && ischar(x)
0023         in = char(x);
0024         if strcmp(in, 'Params')
0025             data_out = plist;
0026             return
0027         elseif strcmp(in, 'Version')
0028             data_out = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 if nargin ~= 3
0035     error (['### please call this function with three parameter: ' ...
0036           'data = set_xy_axis (data, y, x);']);
0037 end
0038
0039 data_out = data;
0040 data_out.x = x;
0041 data_out.y = y;
0042
0043 if nargin == 0
0044     error('### please use: data = set_xy_axis (data, x, y);');
0045 end
0046
0047 % END

```

[Home](#) > [classes](#) > [@xydata](#) > xydata.m

xydata

PURPOSE

XYDATA X-Y data object class constructor.

SYNOPSIS

function xy = xydata(varargin)

DESCRIPTION

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

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [xmlparse](#) XMLPARSE creates a xydata from a xydata xml file.
- [xydata](#) XYDATA X-Y data object class constructor.

This function is called by:

- **xydata** XYDATA X-Y data object class constructor.

SUBFUNCTIONS

- **function xy = init()**

SOURCE CODE

```

0001 function xy = xydata(varargin)
0002 % XYDATA X-Y data object class constructor.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XYDATA X-Y data object class constructor.
0007 %             Create an X-Y data object.
0008 %
0009 %     Properties:
0010 %         name      - name of time-series object
0011 %         x         - x samples vector
0012 %         y         - y samples vector
0013 %         xunits    - units to interpret the time samples (e.g., seconds)
0014 %         yunits    - units to interpret the data samples (e.g., Volts)
0015 %         version   - current version of the constructor
0016 %         created   - current date and time as an number
0017 %
0018 %     Possible constructors:
0019 %         xy = xydata()      - creates a blank xy-data object
0020 %         xy = xydata(y)    - creates an xy data object with the given
0021 %                             y-data.
0022 %         xy = xydata(x,y)  - creates an xy-data object with the given
0023 %                             (x,y)-data.
0024 %
0025 % HISTORY: 30-01-2007 Hewitson
0026 %             Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 VERSION = '$Id: xydata.m,v 1.12 2007/12/24 19:49:09 hewitson Exp $';
0031
0032 %%%% Is this a 'Params' call? %%%%
0033 if nargin == 2 && ischar(varargin{2})
0034     if strcmp(varargin{2}, 'Params')
0035         xy = plist();
0036         return
0037     elseif strcmp(varargin{2}, 'Version')
0038         xy = VERSION;
0039         return
0040     end

```

```

0041 end
0042
0043 %%%%%%%%%% define xydata properties %%%%%%%%%%
0044
0045 function xy = init()
0046     xy.name      = 'None';
0047     xy.x         = [];
0048     xy.y         = [];
0049     xy.xunits    = '';
0050     xy.yunits    = '';
0051     xy.version   = VERSION;
0052     xy.created   = time;
0053     xy           = class(xy, 'xydata');
0054 end
0055
0056 %%%%%%%%%% Create xydata object %%%%%%%%%%
0057
0058 %%%%%%%%%% xy = xydata() %%%%%%%%%%
0059 % create default xydata object
0060 if nargin == 0
0061     xy = init();
0062
0063 elseif nargin == 1
0064
0065     %%%%%%%%%% Create from XML fragment %%%%%%%%%%
0066     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0067         xy = fromxml(varargin{1});
0068     %%%%%%%%%% From File %%%%%%%%%%
0069     elseif ischar(varargin{1})
0070
0071         filename = varargin{1};
0072         [path, name, ext, vers] = fileparts(filename);
0073         switch ext
0074             case '.mat'
0075                 xy = load(filename);
0076                 xy = xy.a;
0077             case '.xml'
0078                 xy = xmlparse(xydata, filename);
0079             otherwise
0080                 error('### Unknown file type.');
```

```

0093     field = fields{ii};
0094
0095     %%% created -> time-object
0096     if strcmp(field, 'created')
0097         created = varargin{1}.created;
0098         if isstruct(created)
0099             created = time(created);
0100         end
0101         xy.created = created;
0102     %%% All other
0103     else
0104         try
0105             xy.(field) = varargin{1}.(field);
0106         catch
0107             error('### The field '%s' in the struct is not a xydata property.',
0108 field)
0109         end
0110     end
0111
0112     %%%%%%%%%%% xy = xydata(y_vector) %%%%%%%%%%%
0113     else
0114         xy = init();
0115         xy.x = 1:length(varargin{1});
0116         xy.y = varargin{1};
0117     end
0118
0119     %%%%%%%%%%% xy = xydata(x_vector, y_vector) %%%%%%%%%%%
0120 elseif nargin == 2
0121
0122     %%%%%%%%%%% From DATABASE
0123     if isa(varargin{1}, 'database')
0124         xy = retrieve(varargin{1}, varargin{2:end});
0125         return
0126     else
0127         xy = init();
0128         xy.x = varargin{1};
0129         xy.y = varargin{2};
0130     end
0131 else
0132     error('### Unknown number of constructor arguments.');
```

```

0133 end
0134
0135
0136 %%%%%%%%%%% Normalize the the x- and y-vector %%%%%%%%%%%
0137 if size(xy.y,2) > size(xy.y,1)
0138     xy.y = [xy.y].';
0139 end
0140 xy.x = reshape(xy.x, size(xy.y));
0141
0142
0143 end % function xy = xydata(varargin)
0144

```


0145

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@miir](#) > [private](#) > [filload.m](#)

filload

PURPOSE

Load a LISO *_iir.fil file to get the filter taps and return a

SYNOPSIS

function f = filload(filename)

DESCRIPTION

Load a LISO *_iir.fil file to get the filter taps and return a miir type object.

M Hewitson 11-02-07

\$Id: filload.m,v 1.4 2007/11/11 16:50:38 hewitson Exp \$

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function f = filload(filename)
0002
0003 % Load a LISO *_iir.fil file to get the filter taps and return a
0004 % miir type object.
0005 %
0006 % M Hewitson 11-02-07
0007 %
0008 % $Id: filload.m,v 1.4 2007/11/11 16:50:38 hewitson Exp $
```

```
0009 %
0010
0011 delims = ' \n';
0012 in = fopen(filename, 'rt');
0013
0014 name = filename(1:end-4);
0015
0016 if(in < 0)
0017     error(['can not open file ' filename]);
0018 end
0019 lbuffer = fgets(in);
0020 n = 1;
0021 ncoeffs = 0;
0022 nbcoeffs = 0;
0023 filt.a = [];
0024 filt.b = [];
0025 norminst = '';
0026
0027 while (lbuffer > 0)
0028     if(length(lbuffer) > 1)
0029         % read the first token on this line
0030         [token, r] = strtok (lbuffer, delims);
0031
0032         % Get fs
0033         if(strcmp(token, 'iir'))
0034             [token, r] = strtok (r, delims); % value
0035             fs = str2double(token);
0036         end
0037
0038         % Get a coeffs
0039         if(token(1) == 'a')
0040             [token, r] = strtok (r, delims); % value
0041             ncoeffs = ncoeffs+1;
0042             filt.a(ncoeffs) = str2double(token);
0043         end
0044         % Get b coeffs
0045         if(token(1) == 'b')
0046             [token, r] = strtok (r, delims); % value
0047             nbcoeffs = nbcoeffs+1;
0048             filt.b(nbcoeffs) = str2double(token);
0049         end
0050
0051     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
```

[Home](#) > [m](#) > [timetools](#) > ltpda_utc2gps.m

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

```

0001 function GPS_time=ltpda_utc2gps(UTC_time)
0002 % LTPDA.UTC2GPS Converts UTC time to GPS seconds.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA.UTC2GPS Converts UTC time to GPS seconds.
0007 %               UTC_time can also be an array of UTC times.
0008 %
0009 % CALL:         GPS_time=ltpda.UTC2GPS(UTC_time)
0010 %
0011 % FORMAT:       UTC time format: 'yyy-mm-dd- HH:MM:SS'
0012 %               GPS time format: Seconds since 6. January 1980
0013 %
0014 %   EXAMPLES:   GPS_time=UTC2GPS('2002-07-19 16:00:00')
0015 %               GPS_time=711129613
0016 %
0017 %               GPS_time=UTC2GPS(['2002-07-19 16:00:00';'2001-07-19
16:00:00'])
0018 %               GPS_time=[711129613 ; 679593613]
0019 %
0020 % VERSION:      $Id: ltpda_utc2gps.m,v 1.3 2007/07/16 12:52:21 ingo Exp $
0021 %
0022 % HISTORY:      xx-xx-xxxx Karsten Koetter
0023 %               Creation
0024 %               24-01-2007 M Hewitson.
0025 %               Maintained
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 GPS_Epoch=datetime('01-06-1980 00:00:00')*86400;
0030
0031 [p q]=size(UTC_time);
0032
0033 for i=1:p
0034     CurrUTC=UTC_time(i,:);
0035     % reformt string to stupid matlab format MM-DD-YYY
0036     CurrUTC=strcat(CurrUTC(6:10),'-',CurrUTC(1:4),CurrUTC(11:length(CurrUTC)));
0037     NUM_time=datetime(CurrUTC)*86400;
0038
0039     GPS_time(i)=round(NUM_time-GPS_Epoch+14);
0040 end

```

[Home](#) > [classes](#) > [@fsdata](#) > set_xy_axis.m

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 2007/11/28 13:54:00 ingo 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

```
0001 function data_out = set_xy_axis(data, x, y)
0002 % SET_XY_AXIS set the x-axis and y-axis values.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET_XY_AXIS set the x-axis and y-axis values.
```

```

0007 %
0008 % CALL:      data = set_xy_axis (data,    x, y);
0009 %          ao.data = set_xy_axis (ao.data, x, y);
0010 %
0011 % VERSION: $Id: set_xy_axis.m,v 1.4 2007/11/28 13:54:00 ingo Exp $
0012 %
0013 % HISTORY: 21-06-2007 Diepholz
0014 %          Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: set_xy_axis.m,v 1.4 2007/11/28 13:54:00 ingo Exp $';
0019
0020 % Check if this is a call for parameters
0021 if nargin == 2
0022     if isa(data, 'fsdata') && ischar(x)
0023         in = char(x);
0024         if strcmp(in, 'Params')
0025             data_out = plist;
0026             return
0027         elseif strcmp(in, 'Version')
0028             data_out = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 if nargin ~= 3
0035     error (['### please call this function with three parameter: ' ...
0036           'data = set_xy_axis (data, x, y);']);
0037 end
0038
0039 if size(x,2) > size(x,1)
0040     x = x.';
0041 end
0042 if size(y,2) > size(y,1)
0043     y = y.';
0044 end
0045
0046 data_out    = data;
0047 data_out.x  = x;
0048 data_out.y  = y;
0049
0050 if nargin == 0
0051     error('### please use: data = set_xy_axis (data, x, y);');
0052 end
0053
0054 % END

```


xyzdata

PURPOSE

XZYDATA X-Y-Z data object class constructor.

SYNOPSIS

function varargout = xyzdata(varargin)

DESCRIPTION

```
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

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

CROSS-REFERENCE INFORMATION

This function calls:

- [xyzdata](#) XZYDATA X-Y-Z data object class constructor.

This function is called by:

- [xyzdata](#) XZYDATA X-Y-Z data object class constructor.

SUBFUNCTIONS

- [function xyz = init\(\)](#)

SOURCE CODE

```

0001 function varargout = xyzdata(varargin)
0002 % XZYDATA X-Y-Z data object class constructor.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: XZYDATA X-Y-Z data object class constructor.
0007 %             Create an X-Y-Z data object.
0008 %
0009 %     Properties:
0010 %         name      - name of time-series object
0011 %         x          - x samples vector
0012 %         y          - y samples vector
0013 %         z          - z samples vector
0014 %         xunits     - units to interpret the time samples (e.g., seconds)
0015 %         yunits     - units to interpret the data samples (e.g., frequency)
0016 %         zunits     - units to interpret the data samples (e.g., Volts)
0017 %         version    - current version of the constructor
0018 %         created    - current date and time as an number
0019 %
0020 %     Possible constructors:
0021 %         xyz = xyzdata()      - creates a blank xyz object
0022 %         xyz = xyzdata(y)    - creates an xyz object with the given
0023 %                               z-data.
0024 %         xyz = xydata(x,y,z) - creates an xyz object with the given
0025 %                               (x,y,z)-data.
0026 %
0027 % HISTORY: 24-12-2007 Hewitson
0028 %         Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 VERSION = '$Id: xyzdata.m,v 1.1 2007/12/24 19:47:06 hewitson Exp $';
0033
0034 %%%% Is this a 'Params' call? %%%%
0035 if nargin == 1 && ischar(varargin{1})
0036     if strcmp(varargin{1}, 'Params')
0037         varargout{1} = plist();
0038         return
0039     elseif strcmp(varargin{1}, 'Version')

```

```

0040     varargout{1} = VERSION;
0041     return
0042 end
0043 end
0044
0045 %%%%%%%%%% define xydata properties %%%%%%%%%%
0046
0047 function xyz = init()
0048     xyz.name      = 'None';
0049     xyz.x         = [];
0050     xyz.y         = [];
0051     xyz.z         = [];
0052     xyz.xunits    = '';
0053     xyz.yunits    = '';
0054     xyz.zunits    = '';
0055     xyz.version   = VERSION;
0056     xyz.created   = time;
0057     xyz          = class(xyz, 'xyzdata');
0058 end
0059
0060 %%%%%%%%%% Create xydata object %%%%%%%%%%
0061
0062 %%%%%%%%%% xy = xydata() %%%%%%%%%%
0063 % create default xydata object
0064 if nargin == 0
0065     xyz = init();
0066
0067 elseif nargin == 1
0068
0069     %%%%%%%%%% Create from XML fragment %%%%%%%%%%
0070     if isa(varargin{1}, 'org.apache.xerces.dom.DeferredElementImpl')
0071         xyz = fromxml(varargin{1});
0072     %%%%%%%%%% From File %%%%%%%%%%
0073     elseif ischar(varargin{1})
0074
0075         filename = varargin{1};
0076         [path, name, ext, vers] = fileparts(filename);
0077         switch ext
0078             case '.mat'
0079                 xyz = load(filename);
0080                 xyz = xyz.a;
0081             case '.xml'
0082                 xyz = xmlparse(xyzdata, filename);
0083             otherwise
0084                 error('### Unknown file type.');
```

```

0092
0093     xyz = init();
0094
0095     fields = fieldnames(varargin{1});
0096     for ii = 1:length(fields)
0097         field = fields{ii};
0098
0099         %%% created -> time-object
0100         if strcmp(field, 'created')
0101             created = varargin{1}.created;
0102             if isstruct(created)
0103                 created = time(created);
0104             end
0105             xyz.created = created;
0106         %%% All other
0107         else
0108             try
0109                 xyz.(field) = varargin{1}.(field);
0110             catch
0111                 error('### The field '%s'' in the struct is not a xydata property.',
field)
0112             end
0113         end
0114     end
0115
0116     %%%%%%%%%% xyz = xyzdata(y_vector) %%%%%%%%%%
0117     else
0118         xyz = init();
0119         xyz.x = 1:size(varargin{1}, 1);
0120         xyz.y = 1:size(varargin{1}, 2);
0121         xyz.z = varargin{1};
0122     end
0123
0124     %%%%%%%%%% xy = xydata(x_vector, y_vector) %%%%%%%%%%
0125 elseif nargin == 2
0126
0127     %%%%%%%%%% From DATABASE
0128     if isa(varargin{1}, 'database')
0129         xyz = retrieve(varargin{1}, varargin{2:end});
0130     else
0131         error('### Unknown constructor.');
```

```
0143 %%%%%%%%%%    Normalize the the x- and y-vector    %%%%%%%%%%%%%%
0144
0145 nx = size(xyz.z, 2);
0146 ny = size(xyz.z, 1);
0147
0148 if length(xyz.x) ~= nx
0149     error('### X-vector has wrong length. ');
0150 end
0151 if length(xyz.y) ~= ny
0152     error('### Y-vector has wrong length. ');
0153 end
0154 if size(xyz.y,2) > size(xyz.y,1)
0155     xyz.y = [xyz.y].';
0156 end
0157 if size(xyz.x,2) > size(xyz.x,1)
0158     xyz.x = [xyz.x].';
0159 end
0160
0161
0162 %%%% Set output
0163 varargout{1} = xyz;
0164
0165 end % function xy = xydata(varargin)
0166
0167
```

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

```
0001 function ltpda_xaxis(x1,x2)
0002 % LTPDA_XAXIS Set the X axis range of the current figure
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_XAXIS Set the X axis range of the current figure
0007 %
0008 % CALL:            ltpda_xaxis(x1,x2)
0009 %
```

```
0010 % VERSION:      $Id: ltpda_xaxis.m,v 1.4 2007/07/16 12:52:20 ingo Exp $
0011 %
0012 % HISTORY:       09-10-2001 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 set(gca, 'XLim', [x1 x2]);
0018
0019 % END
```

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 2007/11/28 13:54:01 ingo 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

```
0001 function data_out = set_xy_axis(data, x, y)
0002 % SET_XY_AXIS set the x-axis and y-axis values.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET_XY_AXIS set the x-axis and y-axis values.
```



```

0007 %
0008 % CALL:      data = set_xy_axis (data,    x, y);
0009 %          ao.data = set_xy_axis (ao.data, x, y);
0010 %
0011 % VERSION: $Id: set_xy_axis.m,v 1.4 2007/11/28 13:54:01 ingo Exp $
0012 %
0013 % HISTORY: 21-06-2007 Diepholz
0014 %          Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: set_xy_axis.m,v 1.4 2007/11/28 13:54:01 ingo Exp $';
0019
0020 % Check if this is a call for parameters
0021 if nargin == 2
0022     if isa(data, 'tsdata') && ischar(x)
0023         in = char(x);
0024         if strcmp(in, 'Params')
0025             data_out = plist;
0026             return
0027         elseif strcmp(in, 'Version')
0028             data_out = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 if nargin ~= 3
0035     error (['### please call this function with three parameter: ' ...
0036           'data = set_xy_axis (data, x, y);']);
0037 end
0038
0039 if size(x,2) > size(x,1)
0040     x = x.';
0041 end
0042 if size(y,2) > size(y,1)
0043     y = y.';
0044 end
0045
0046 data_out = data;
0047 data_out.x = x;
0048 data_out.y = y;
0049
0050 if nargin == 0
0051     error('### please use: data = set_xy_axis (data, x, y);');
0052 end
0053
0054 % END

```

[Home](#) > [m](#) > [sigproc](#) > [time_domain](#) > [ltpda_xcorr.m](#)

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
```

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

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SUBFUNCTIONS

- function lmin = findShortestVector(as)
- function fs = findFsMax(as)
- function plo = getDefaultPL()

SOURCE CODE

```
0001 function varargout = ltpda_xcorr(varargin)
0002 % LTPDA_XCORR makes cross-correlation estimates of the time-series
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_XCORR makes cross-correlation estimates of the time-series
0007 %               objects in the input analysis objects. The cross-correlation is
0008 %               computed using MATLAB's xcorr (>> help xcorr).
0009 %
0010 % CALL:         b = ltpda_xcorr(a1,a2,a3,...,pl)
0011 %
0012 % INPUTS:       b      - output analysis objects
0013 %               aN      - input analysis objects (at least two)
0014 %               pl      - input parameter list
0015 %
0016 %               The function makes correlation estimates between a1 and all other
0017 %               input AOs. Therefore, if the input argument list contains N
analysis
0018 %               objects, the output a will contain N-1 CPSD estimates.
0019 %
0020 %               If only on AO is input, the auto-correlation is computed.
0021 %
0022 %               If the last input argument is a parameter list (plist) it is
used.
```

```

0023 %           The following parameters are recognised.
0024 %
0025 % PARAMETERS:  'MaxLag'    - compute over range of lags -MaxLag to MaxLag
[default: M-1]
0026 %           'Scale'      - normalisation of the correlation. Choose from:
0027 %           'biased'     - scales the raw cross-correlation by 1/M.
0028 %           'unbiased'   - scales the raw correlation by 1/(M-abs(lags)).
0029 %           'coeff'      - normalizes the sequence so that the auto-
correlations
0030 %                               at zero lag are identically 1.0.
0031 %           'none'       - no scaling (this is the default).
0032 %
0033 %           M is the length of longest input vector. If one vector is
shorted,
0034 %           it is zero padded.
0035 %
0036 %           The following call returns a parameter list object that contains
0037 %           the default parameter values:
0038 %
0039 %           >> pl = ltpda_xcorr('Params')
0040 %
0041 % VERSION:      $Id: ltpda_xcorr.m,v 1.4 2007/11/08 10:35:04 hewitson Exp $
0042 %
0043 % HISTORY:      07-02-2007 M Hewitson
0044 %                Creation
0045 %
0046 %%%%%%%%%%%%%%%
0047
0048 ALGONAME = mfilename;
0049 VERSION  = '$Id: ltpda_xcorr.m,v 1.4 2007/11/08 10:35:04 hewitson Exp $';
0050
0051 % Check if this is a call for parameters
0052 if nargin == 1
0053     in = char(varargin{1});
0054     if strcmp(in, 'Params')
0055         varargout{1} = getDefaultPL();
0056         return
0057     end
0058 end
0059
0060 % capture input variable names
0061 invars = {};
0062 as      = [];
0063 ps      = [];
0064 for j=1:nargin
0065     invars = [invars cellstr(inputname(j))];
0066     if isa(varargin{j}, 'ao')
0067         as = [as varargin{j}];
0068     end
0069     if isa(varargin{j}, 'plist')
0070         ps = [ps varargin{j}];
0071     end
0072 end

```

```

0073
0074 % check plist
0075 if isempty(ps)
0076     pl = getDefaultPL();
0077 else
0078     pl = combine(ps, getDefaultPL());
0079 end
0080
0081 na = length(as);
0082 if na < 2
0083     error('### LTPDA_XSPEC needs at least two AOs to make a transfer function.');
```

```

0084 end
0085
0086
0087 %----- Resample all AOs
0088
0089 disp('*** Resampling AOs...');
0090 fsmax = findFsMax(as);
0091 fspl = plist(param('fsout', fsmax));
0092 bs = [];
0093 for i=1:na
0094     a = as(i);
0095     % check this is a time-series object
0096     if ~isa(a.data, 'tsdata')
0097         error('### ltpda_xspec requires tsdata (time-series) inputs.');
```

```

0098     end
0099     % Check Fs
0100     if a.data.fs ~= fsmax
0101         warning(sprintf('!!! Resampling AO %s/%s to %f Hz', a.name, a.data.name,
fsmax));
0102         a = resample(a, fspl);
0103     end
0104     bs = [bs a];
0105 end
0106 as = bs;
0107 clear bs;
0108 disp('*** Done.');
```

```

0109
0110
0111 %----- Truncate all vectors
0112
0113 % Get shortest vector
0114 disp('*** Truncating all vectors...');
```

```

0115 lmin = findShortestVector(as);
0116 nsecs = lmin / fsmax;
0117 bs = [];
0118 for i=1:na
0119     a = as(i);
0120     if len(a) ~= lmin
0121         warning(sprintf('!!! Truncating AO %s/%s to %d secs', a.name, a.data.name,
nsecs));
0122         bs = [bs select(a, 1:lmin)];
```

```

0123     else
0124         bs = [bs a];
0125     end
0126 end
0127 as = bs;
0128 clear bs;
0129 disp('*** Done.');
```

```

0130
0131 %----- check input parameters
0132
0133 % Maximum lag for Xcorr
0134 MaxLag = find(pl, 'MaxLag');
0135
0136 % Scale for Xcorr
0137 scale = find(pl, 'Scale');
```

```

0138
0139 % Loop over input AOs
0140 bs = cell(na);
0141 for j=1:na
0142     aj = as(j);
0143     for k=1:na
0144         ak = as(k);
0145         % ----- Make Xspec estimate
0146
0147         % Compute cross-correlation estimates using XCORR
0148         if MaxLag == -1
0149             MaxLag = len(aj);
0150         end
0151         [c,lags] = xcorr(aj.data.y,ak.data.y, MaxLag, scale);
0152
0153         % create new output fsdata
0154         xy = xydata(lags./fsmax, c);
0155         xy = set(xy, 'name', sprintf('xcorr(%s,%s)', aj.data.name, ak.data.name));
0156         xy = set(xy, 'xunits', 's');
0157         xy = set(xy, 'yunits', '');
0158
0159         %----- create new output history
0160
0161         % we need to get the input histories in the same order as the inputs
0162         % to this function call, not in the order of the input to tfestimate;
0163         % otherwise the resulting matrix on a 'create from history' will be
0164         % mirrored.
0165         if j<k
0166             h = history(ALGONAME, VERSION, pl, [aj.hist ak.hist]);
0167         else
0168             h = history(ALGONAME, VERSION, pl, [ak.hist aj.hist]);
0169         end
0170         h = set(h, 'invars', [invars(j) invars(k)]);
0171
0172         % make output analysis object
0173         b = ao(xy, h);
0174
0175         % set name
```

```

0176     if isempty(invars{k})
0177         nk = ak.name;
0178     else
0179         nk = invars{k};
0180     end
0181     if isempty(invars{j})
0182         nj = aj.name;
0183     else
0184         nj = invars{j};
0185     end
0186     b = setnh(b, 'name', sprintf('xcorr(%s->%s)', nj, nk));
0187
0188     % add to outputs
0189     bs(j,k) = {b};
0190 end % End second loop over AOs
0191 end % End first loop over AOs
0192
0193 % now we have a cell matrix of AOs but
0194 % we want a normal matrix
0195 b = [bs{:}];
0196 varargout{1} = reshape(b, na, na);
0197
0198
0199 %-----
0200 % Returns the length of the shortest vector in samples
0201 function lmin = findShortestVector(as)
0202
0203 lmin = 1e20;
0204 for j=1:length(as)
0205     if len(as(j)) < lmin
0206         lmin = len(as(j));
0207     end
0208 end
0209
0210
0211 %-----
0212 % Returns the max Fs of a set of AOs
0213 function fs = findFsMax(as)
0214
0215 fs = 0;
0216 for j=1:length(as)
0217     a = as(j);
0218     if a.data.fs > fs
0219         fs = a.data.fs;
0220     end
0221 end
0222
0223 %-----
0224 % Get default params
0225 function plo = getDefaultPL()
0226
0227 disp('* creating default plist...');

```

```
0228 plo = plist();
0229 plo = append(plo, param('MaxLag', -1));
0230 plo = append(plo, param('Scale', 'none'));
0231 disp('* done. ');
0232
0233
```

Generated on Tue 22-Jan-2008 10:39:13 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@xydata](#) > set_xy_axis.m

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 1.3 2007/11/28 13:54:01 ingo 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

```
0001 function data_out = set_xy_axis(data, x, y)
0002 % SET_XY_AXIS set the x-axis and y-axis values.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: SET_XY_AXIS set the x-axis and y-axis values.
```

```

0007 %
0008 % CALL:      data = set_xy_axis (data,    x, y);
0009 %          ao.data = set_xy_axis (ao.data, x, y);
0010 %
0011 % VERSION: $Id: set_xy_axis.m,v 1.3 2007/11/28 13:54:01 ingo Exp $
0012 %
0013 % HISTORY: 21-06-2007 Diepholz
0014 %          Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: set_xy_axis.m,v 1.3 2007/11/28 13:54:01 ingo Exp $';
0019
0020 % Check if this is a call for parameters
0021 if nargin == 2
0022     if isa(data, 'xydata') && ischar(x)
0023         in = char(x);
0024         if strcmp(in, 'Params')
0025             data_out = plist;
0026             return
0027         elseif strcmp(in, 'Version')
0028             data_out = VERSION;
0029             return
0030         end
0031     end
0032 end
0033
0034 if nargin ~= 3
0035     error (['### please call this function with three parameter: ' ...
0036           'data = set_xy_axis (data, x, y);']);
0037 end
0038
0039 if size(x,2) > size(x,1)
0040     x = x.';
0041 end
0042 if size(y,2) > size(y,1)
0043     y = y.';
0044 end
0045
0046 data_out = data;
0047 data_out.x = x;
0048 data_out.y = y;
0049
0050 if nargin == 0
0051     error('### please use: data = set_xy_axis (data, x, y);');
0052 end
0053
0054 % END

```

[Home](#) > [m](#) > [helper](#) > ltpda_xml_read_datasamples.m

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

```
0001 function data = ltpda_xml_read_datasamples(node, dType)
0002
0003 % LTPDA_XML_READ_DATASAMPLES read data samples from the given node of an XML
0004 % file.
0005 % M Hewitson 19-02-07
0006 %
0007 % $Id: ltpda_xml_read_datasamples.m,v 1.5 2007/10/26 14:39:54 hewitson Exp $
0008 %
0009
0010
0011
0012 switch dType
0013     case 'real'
0014
0015         disp('-- reading real data samples');
0016
```

```

0017     data.x = [];
0018     data.y = [];
0019
0020     children = node.getChildNodes;
0021     for j=1:children.getLength
0022         ch = children.item(j-1);
0023         nodeName = char(ch.getNodeName);
0024         cont = char(ch.getTextContent);
0025         switch nodeName
0026             case 'xset'
0027                 vals = sscanf(cont, '%g ');
0028                 data.x = [data.x ; vals];
0029                 disp(['* read ' num2str(length(data.x)) ' X samples']);
0030             case 'yset'
0031                 vals = sscanf(cont, '%g ');
0032                 data.y = [data.y ; vals];
0033                 disp(['* read ' num2str(length(data.y)) ' Y samples']);
0034             otherwise
0035                 % warning(['!!! unknown data sample child node.' nodeName]);
0036         end
0037     end
0038
0039 %     data.x = data.x';
0040 %     data.y = data.y';
0041
0042     case 'complex'
0043
0044         disp('-- reading complex data samples');
0045
0046         data.x = [];
0047         ry     = [];
0048         iy     = [];
0049
0050         children = node.getChildNodes;
0051         for j=1:children.getLength
0052             ch = children.item(j-1);
0053             nodeName = char(ch.getNodeName);
0054             cont = char(ch.getTextContent);
0055             switch nodeName
0056                 case 'xset'
0057                     vals = sscanf(cont, '%f ');
0058                     data.x = [data.x ; vals];
0059                     disp(['* read ' num2str(length(data.x)) ' X samples']);
0060                 case 'ryset'
0061                     vals = sscanf(cont, '%f ');
0062                     ry = [ry ; vals];
0063                     disp(['* read ' num2str(length(ry)) ' real Y samples']);
0064                 case 'iyset'
0065                     vals = sscanf(cont, '%f ');
0066                     iy = [iy ; vals];
0067                     disp(['* read ' num2str(length(iy)) ' imag Y samples']);

```

```
0068         otherwise
0069             %         warning(['!!! unknown data sample child node.' nodeName]);
0070         end
0071     end
0072
0073 %     data.x = data.x';
0074 %     ry     = ry';
0075 %     iy     = iy';
0076
0077     data.y = complex(ry, iy);
0078
0079     otherwise
0080         error('### unknown data type. ');
0081 end
0082
0083
0084 % END
0085
```

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

```
0001 function ltpda_xscale(scale)
0002 % LTPDA_XSCALE Set the X scale of the current axis
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_XSCALE Set the X scale of the current axis
0007 %
0008 % CALL:            ltpda_xscale('scale')            scale = 'lin' or 'log';
0009 %
```

```
0010 % VERSION:      $Id: ltpda_xscale.m,v 1.2 2007/07/16 12:52:20 ingo Exp $
0011 %
0012 % HISTORY:       02-05-2004 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 set(gca, 'XScale', scale);
0018
0019 % END
```