

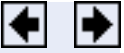






--	--





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](#)

[Trouble-shooting](#) 

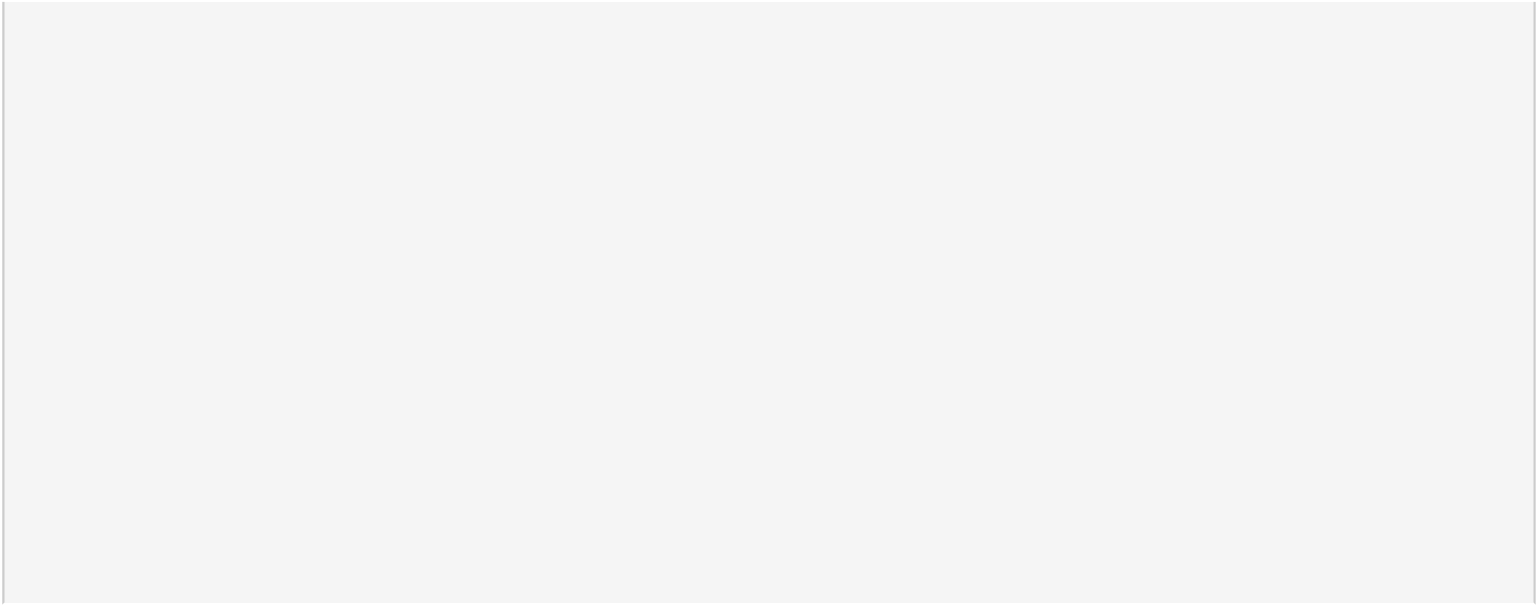
©LTP Team





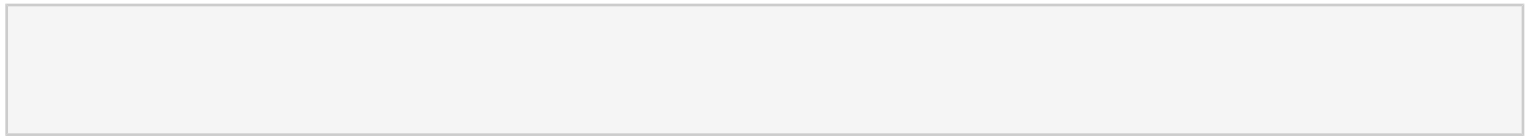
Examples

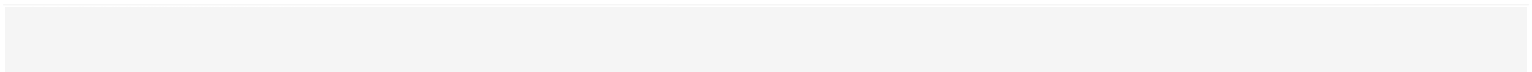
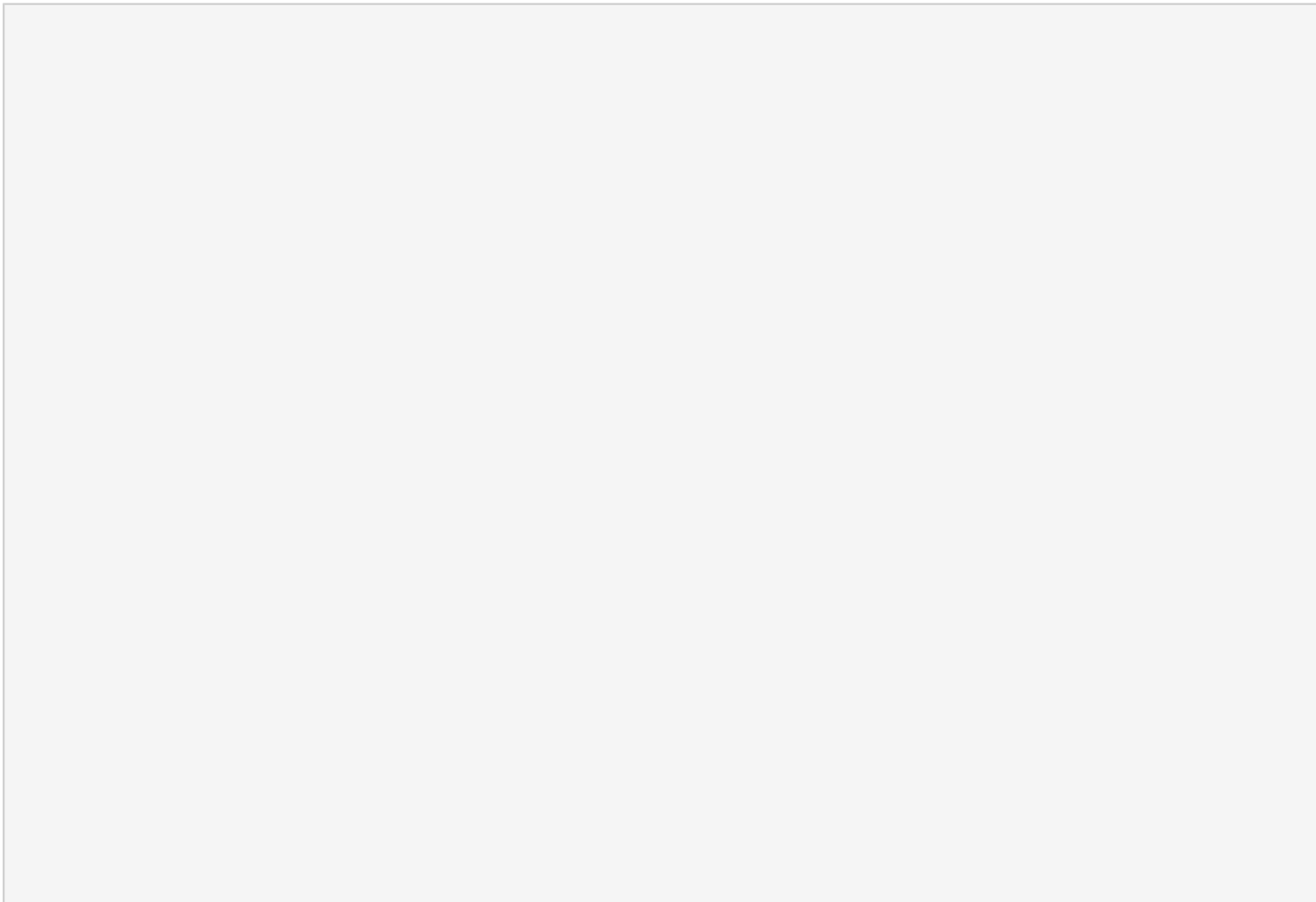




LTPDA Toolbox

[contents](#)





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 parameter lists directly.

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

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

Appending parameters to a parameter list.

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

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

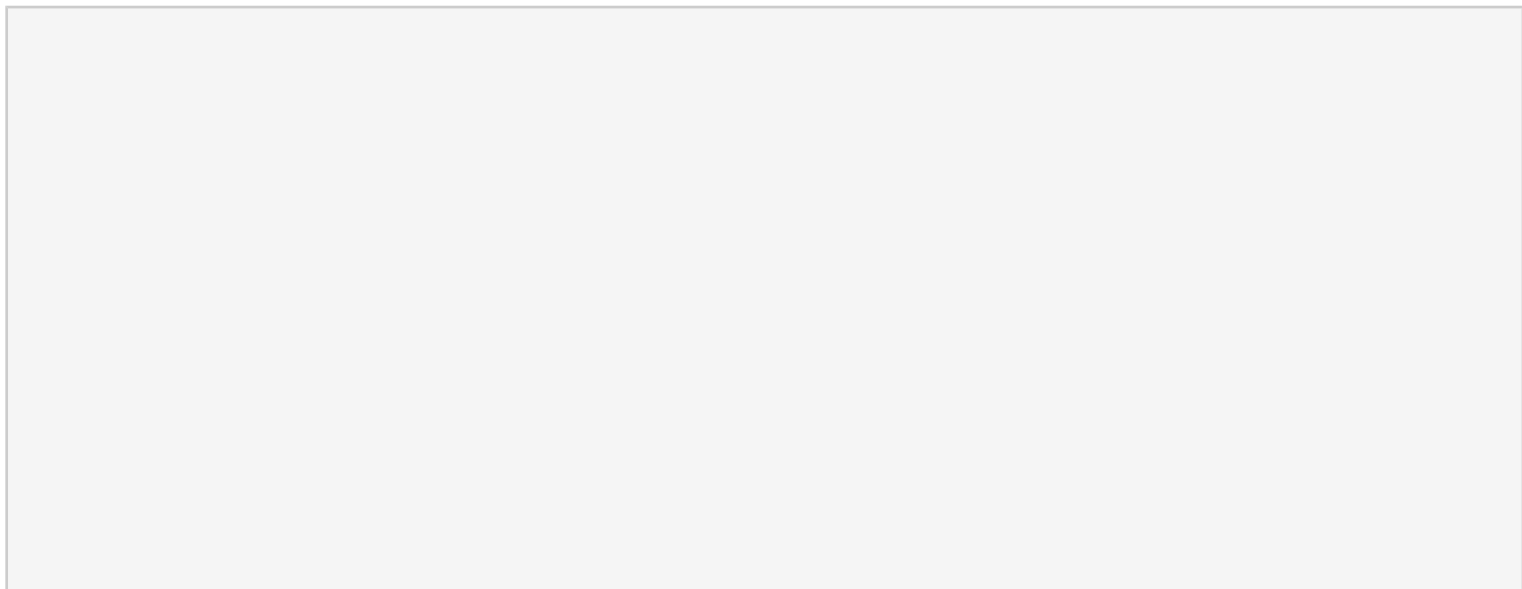
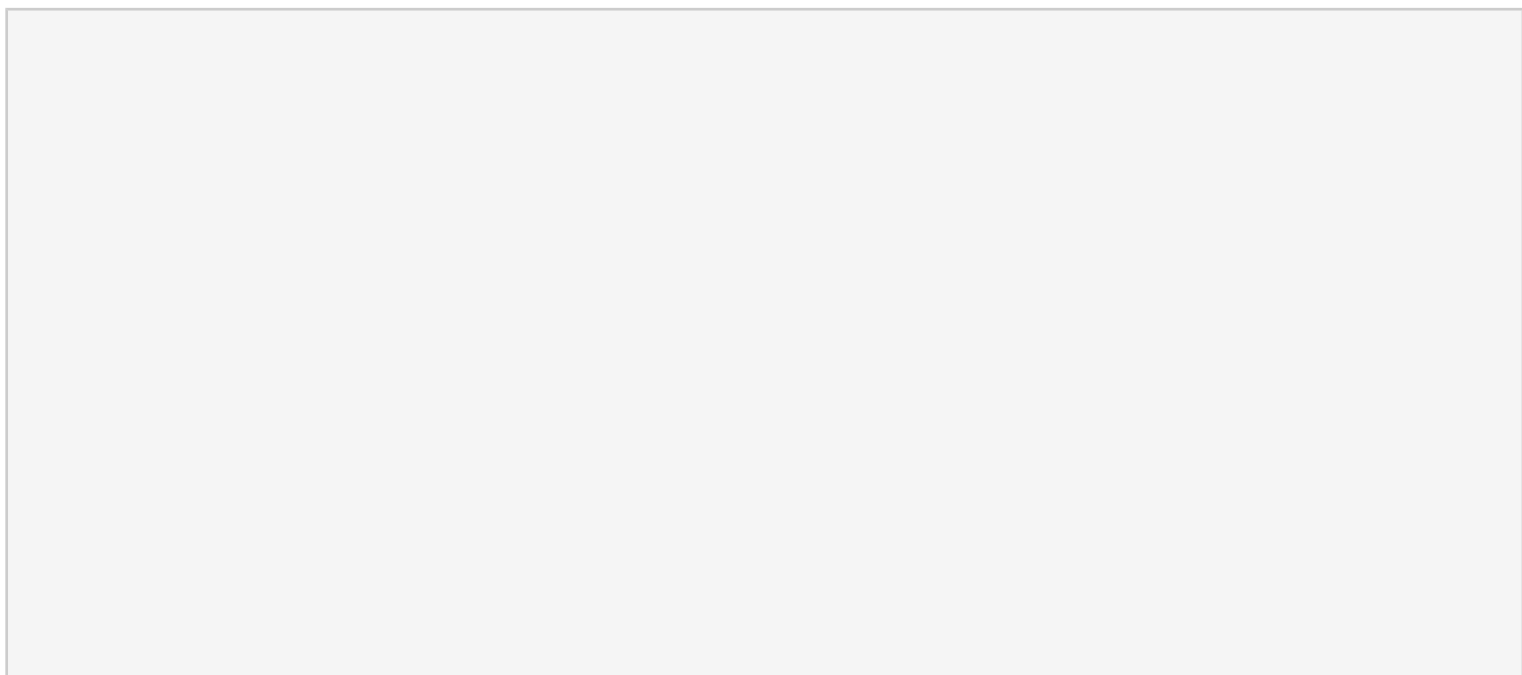

What are LTPDA spectral windows? (LTPDA Toolbox)

What are LTPDA spectral windows? (LTPDA Toolbox)



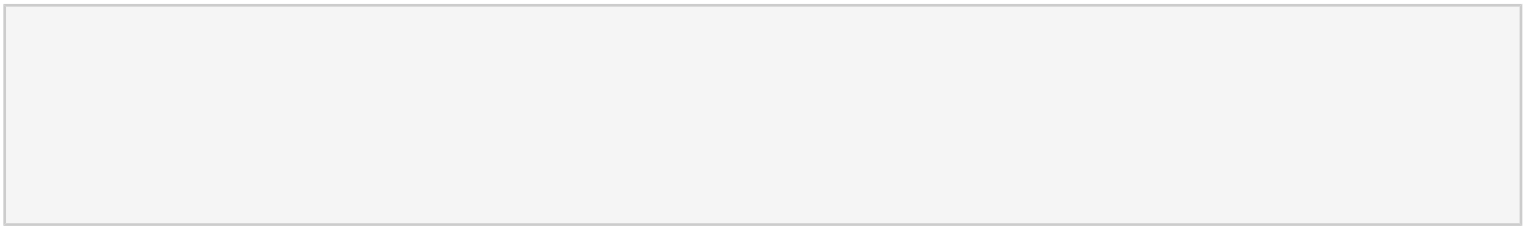
Create spectral windows

To create a spectral window object, you call the `specwin` class constructor. The following code







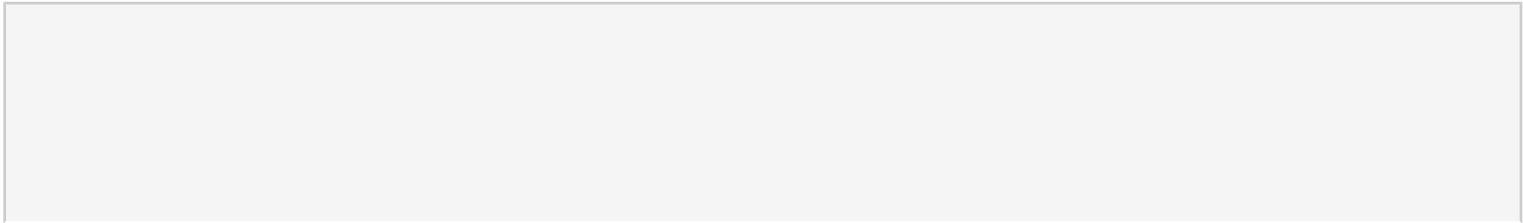


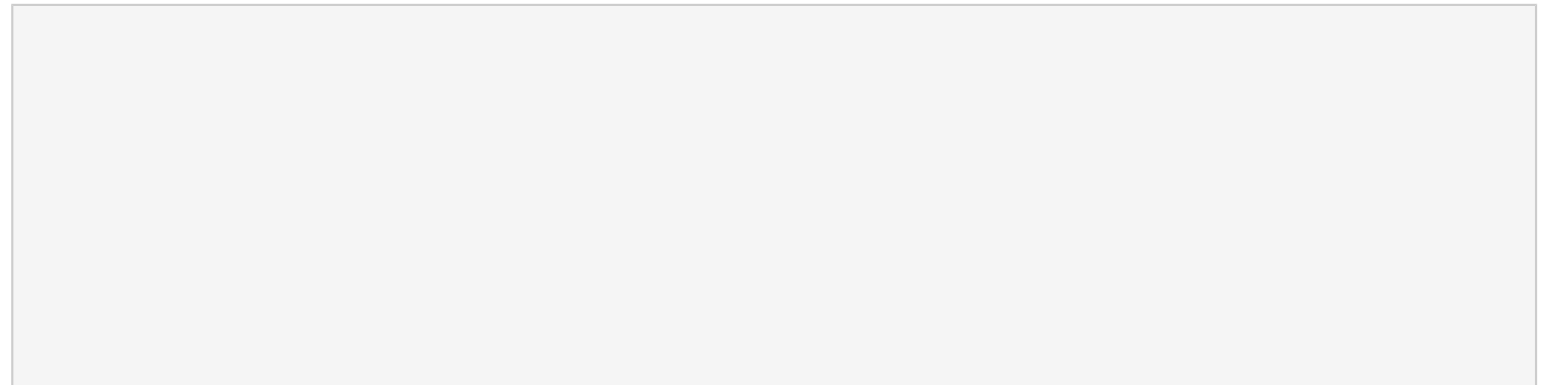
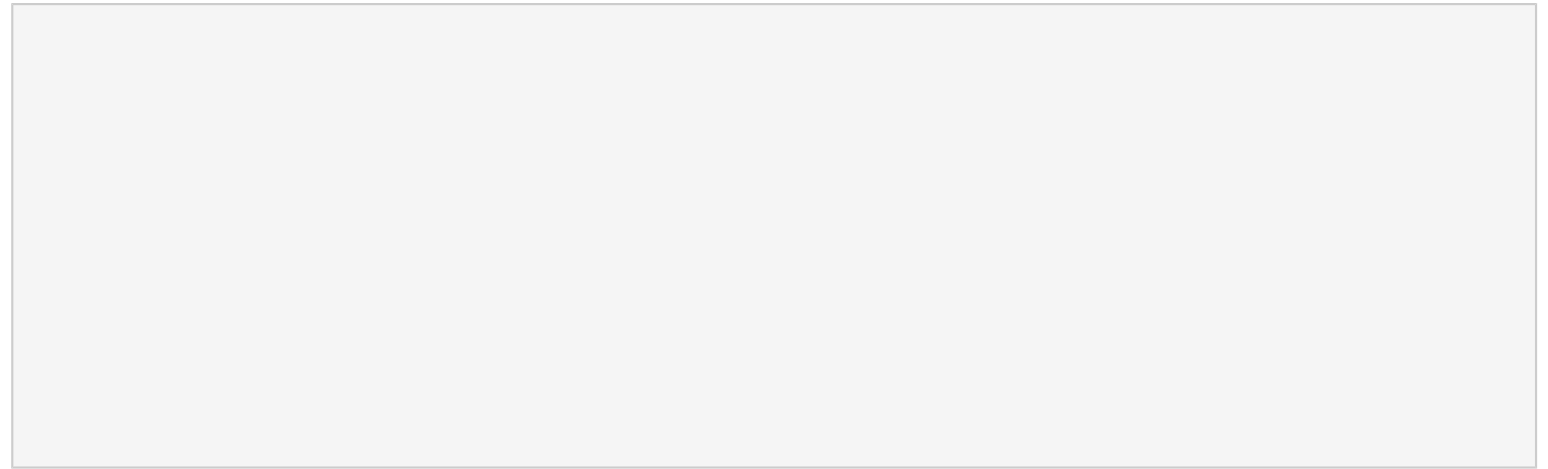


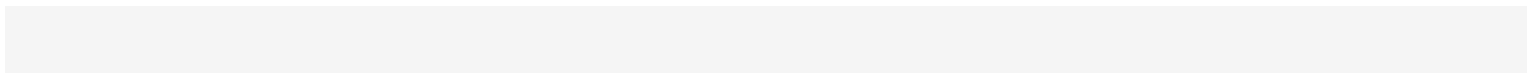
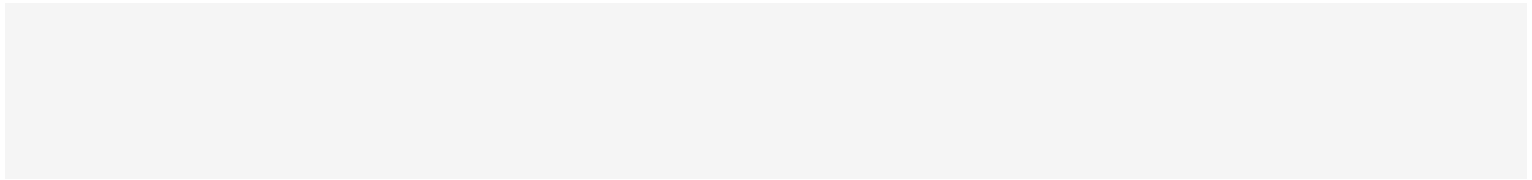
LTPDA Toolbox

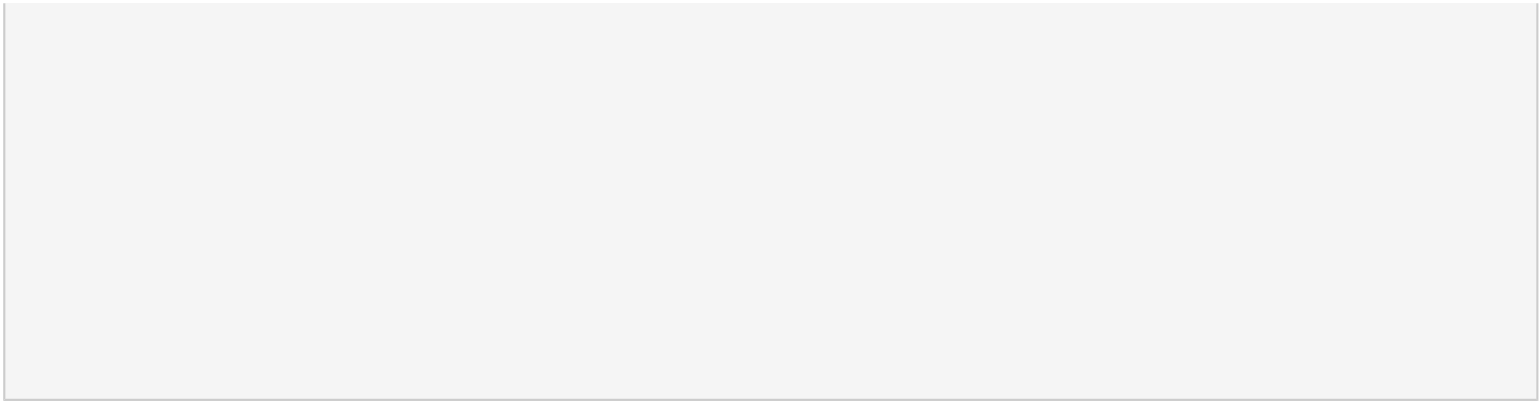
[contents](#)











LTPDA Toolbox



...



 Graphical User Interfaces in LTPDA

The LTPDA Analysis GUI 

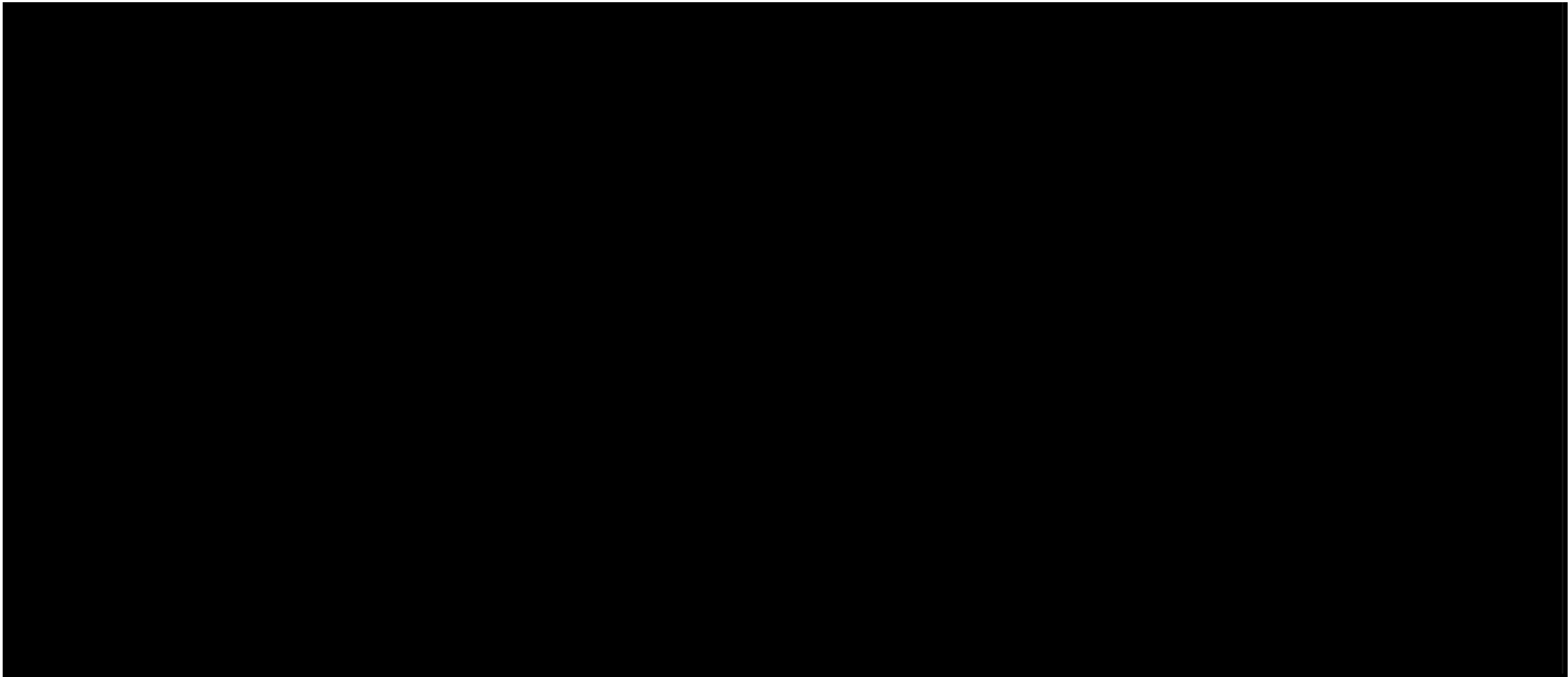
©LTP Team

LTPDA Toolbox

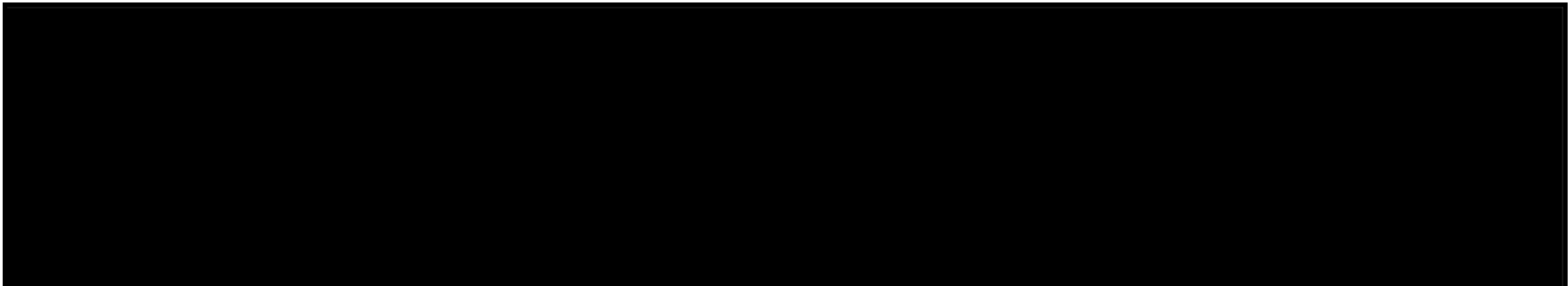
[contents](#)

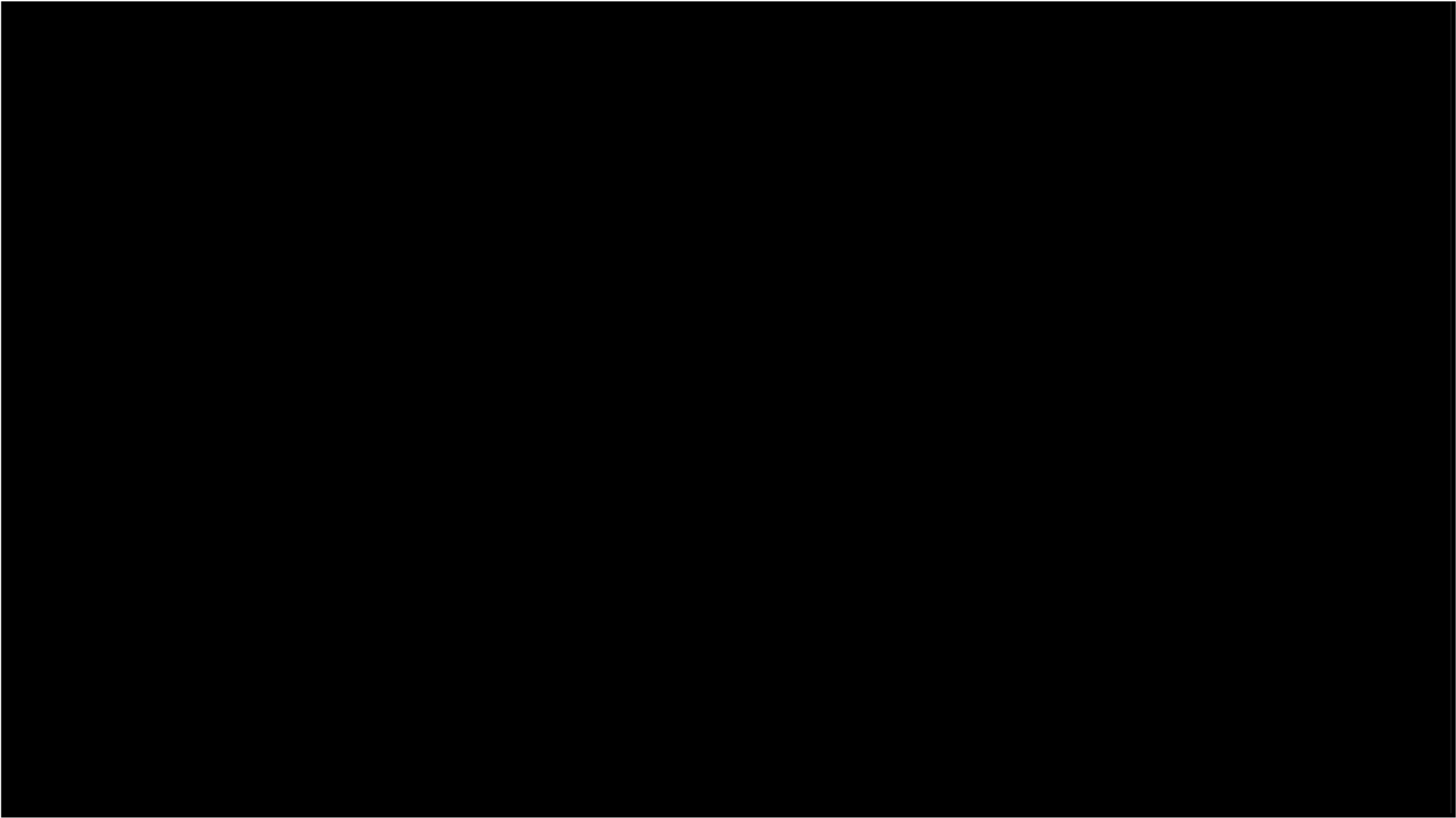




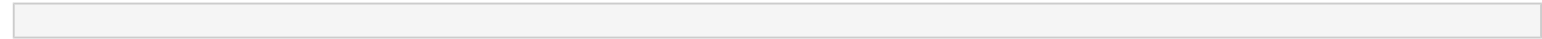


You can add poles and zeros to the model by entering the frequency (and Q) in the edit boxes, then click **add pole** or **add zero** as appropriate. The **respons/** is then updated in the **respons/** axes.





LTPDA Toolbox







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

 The constructor helper



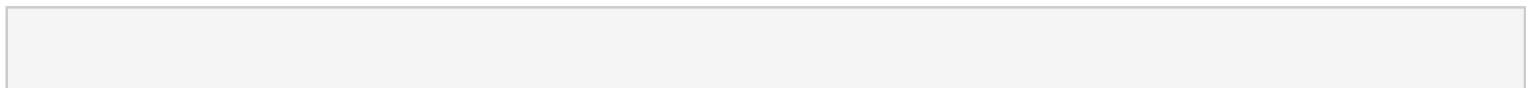
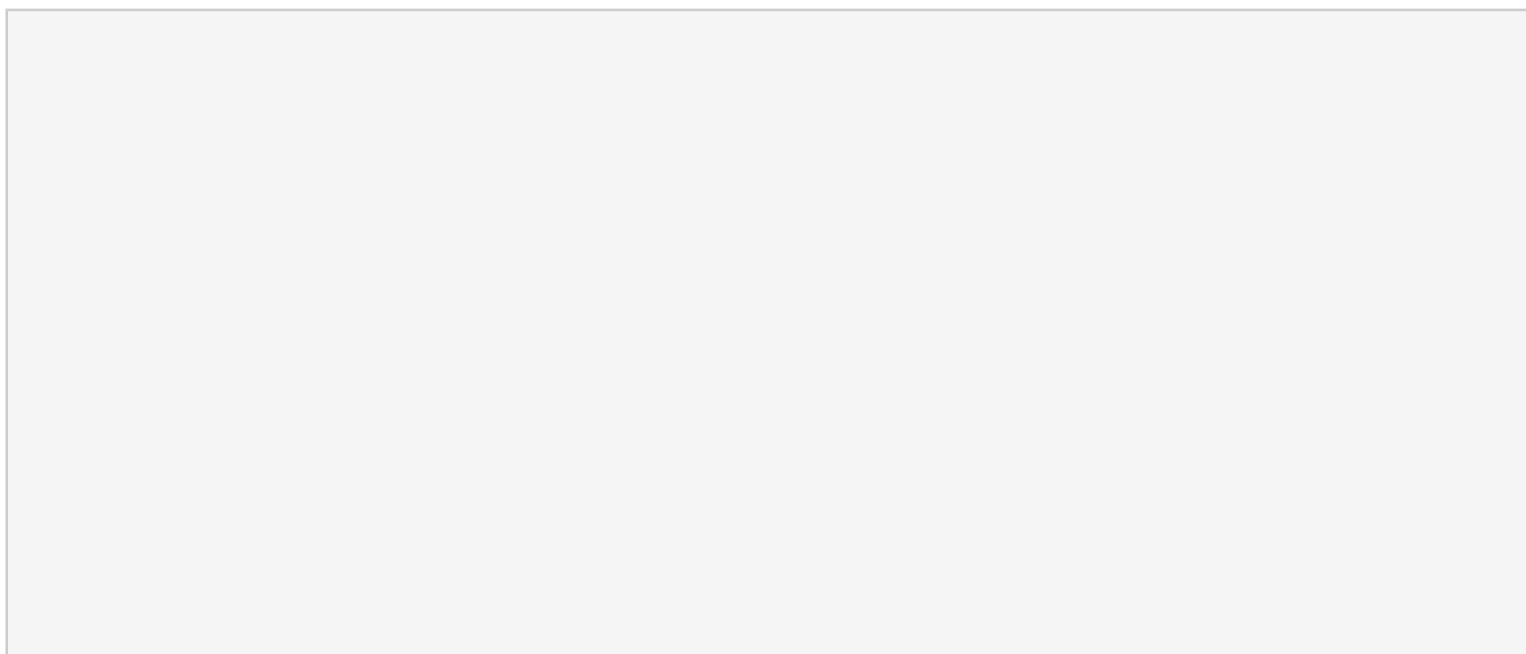
The object list is filled with all LTPDA User Objects currently in the MATLAB workspace.

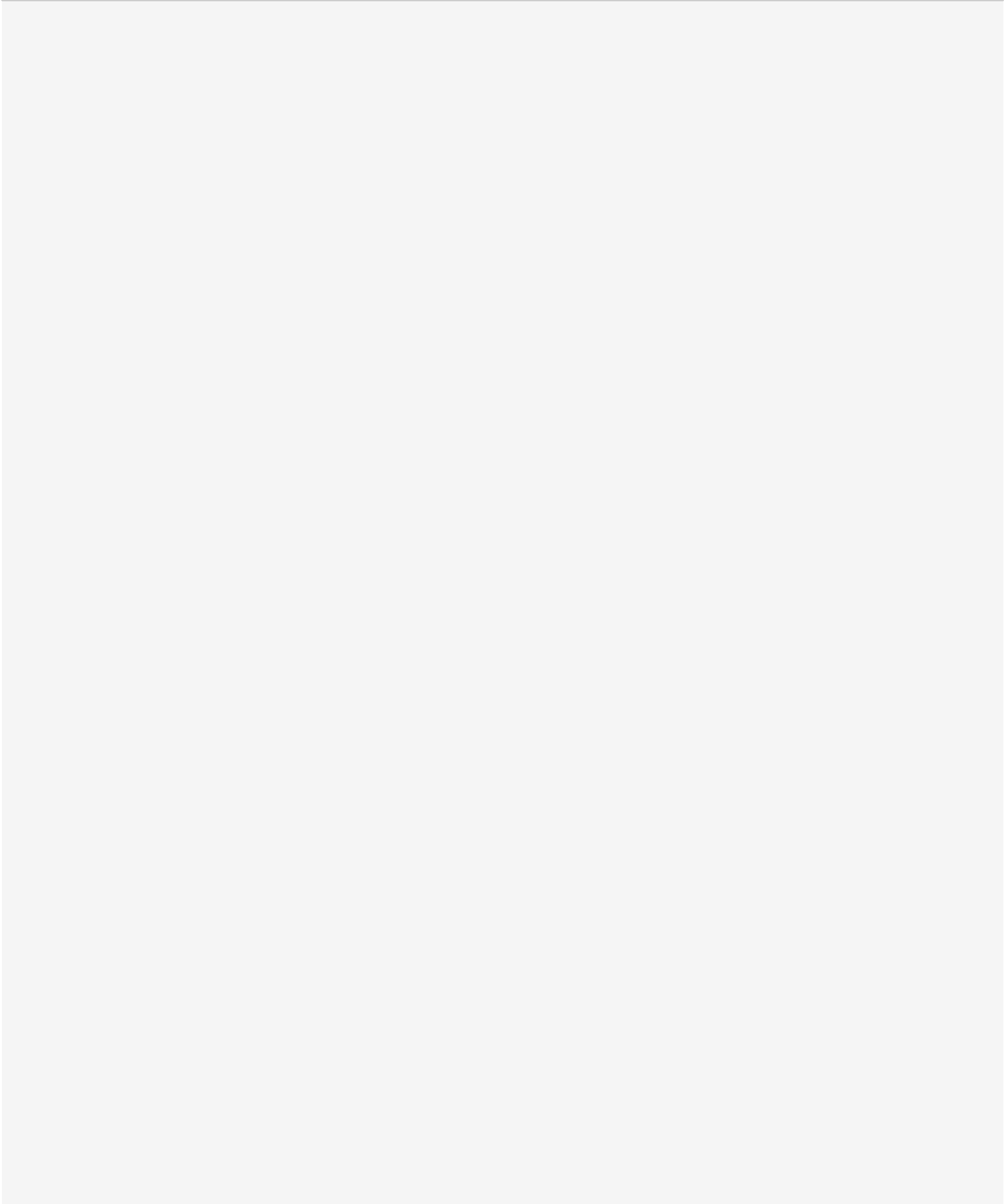




As you can see, each object that is submitted to a repository receives a unique ID number. This ID number is used to link together the various pieces of meta-data that are collected about each object. In addition, each object that is submitted is check-summed using the [MD5 algorithm](#). That way, the

Connecting to an LTPDA Repository






```
% 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 transact      6where '...
      'transdate>"2007-11-14 00:00:00" and '...
      'transdate<"2007-11-15 00:00:00"'];
info = ltpda_dbquery(server, dbname, q)
```




LTPDA Toolbo



©LTP Team

ao Class

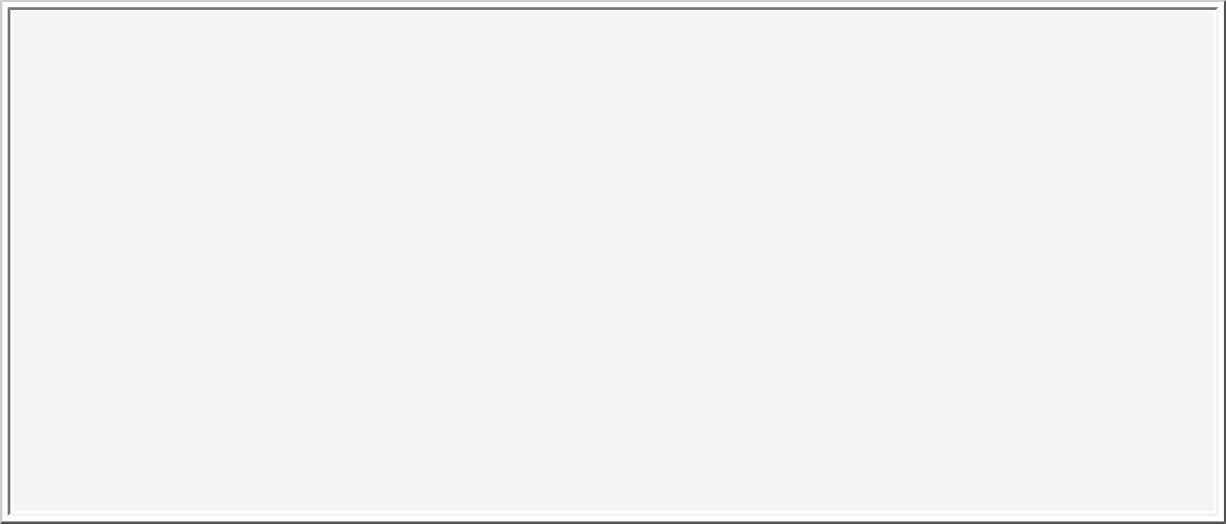
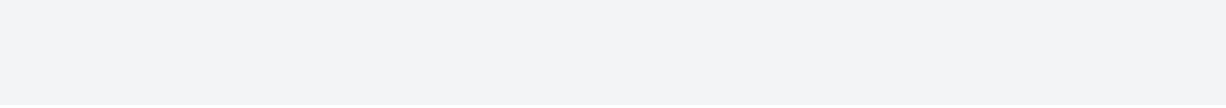
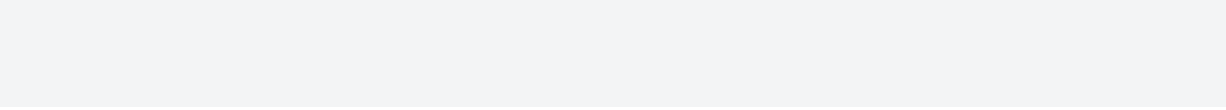
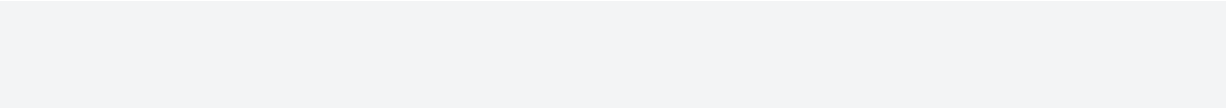
Properties	DDDDDDDDDD
Methods	DDDDDDDDDD
Constructor Examples	

--	--

--	--

--	--

data	Data object associated with this AO
hist	History object associated with this AO
provenance	Provenance object associated with this AO













version	
created	











<code>winlen</code>	Window length
<code>win</code>	Window samples (column vector)
<code>plist</code>	The parameter list object which creates the SPECWIN object
<code>version</code>	CVS version string of the constructor
<code>created</code>	Time object which stores the creation time of the SPECWIN object

[Class methods](#)

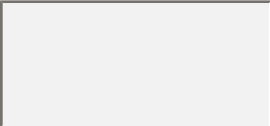




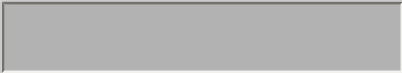


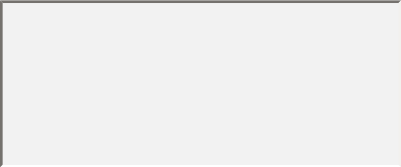


--	--



--	--





--	--

 zero Class

Functions - Alphabetical List 

©LTP Team

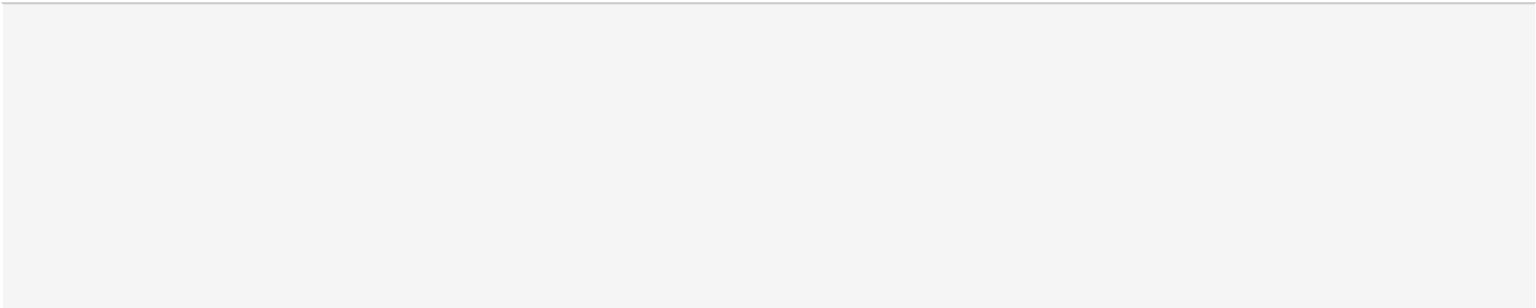
Additional parameters:

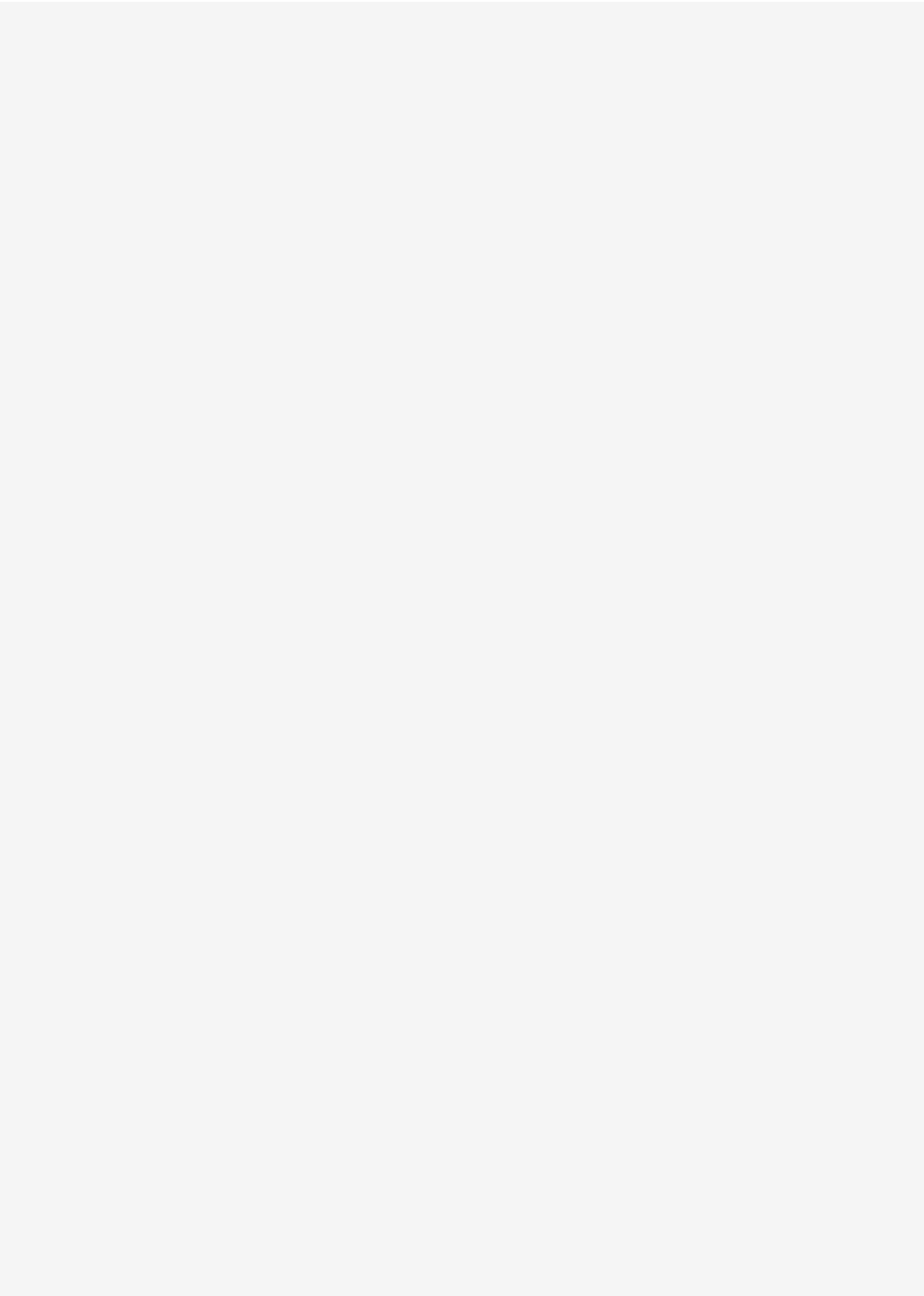
From Polynomial

```
Example: a = ao(plist('polyval', [1 2 3], 'Nsecs', 10, 'fs', 10);
```

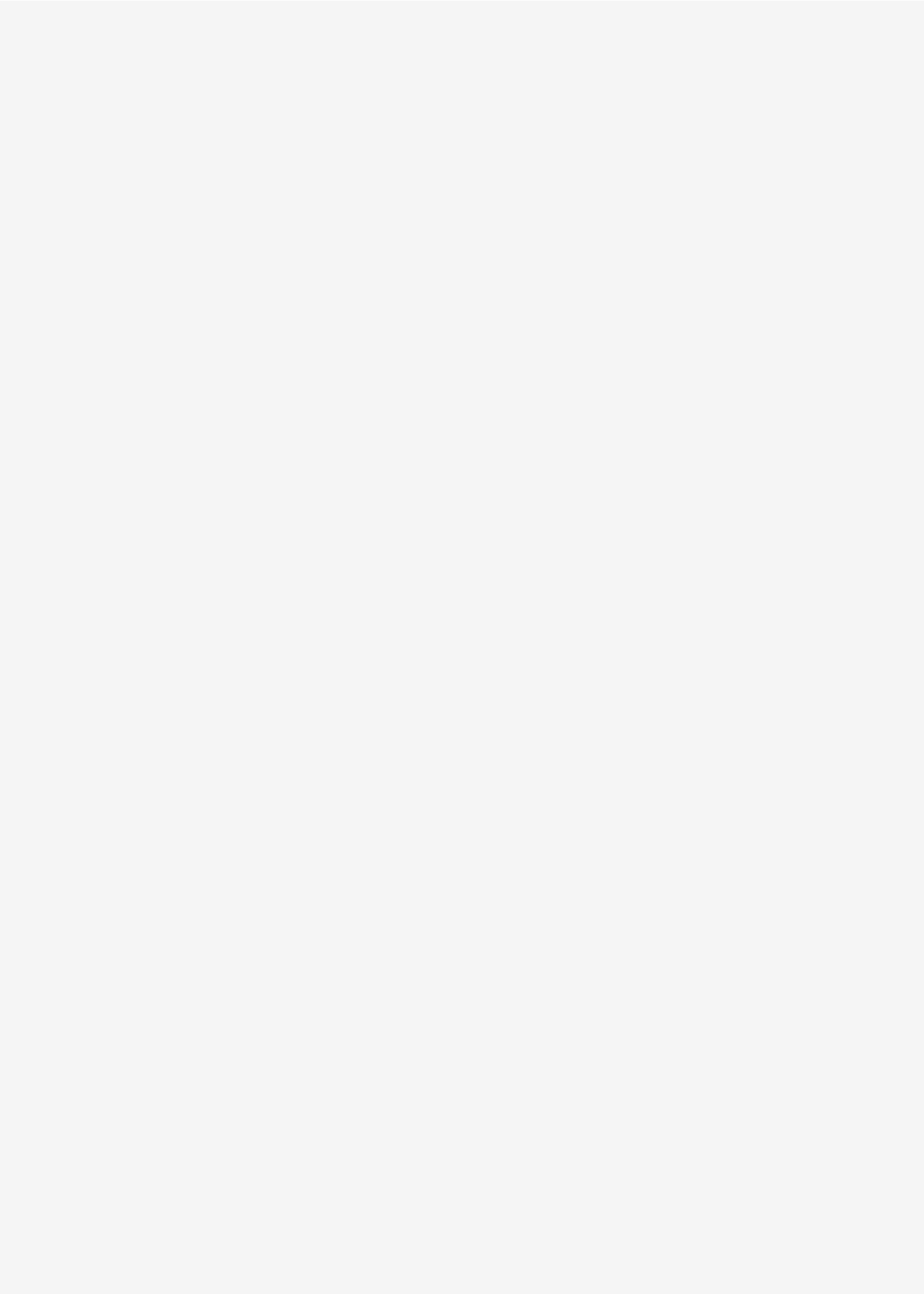
From Plist

```
p = plist('waveform', 'noise', 'fs', 10, 'nsecs', 1000);
```

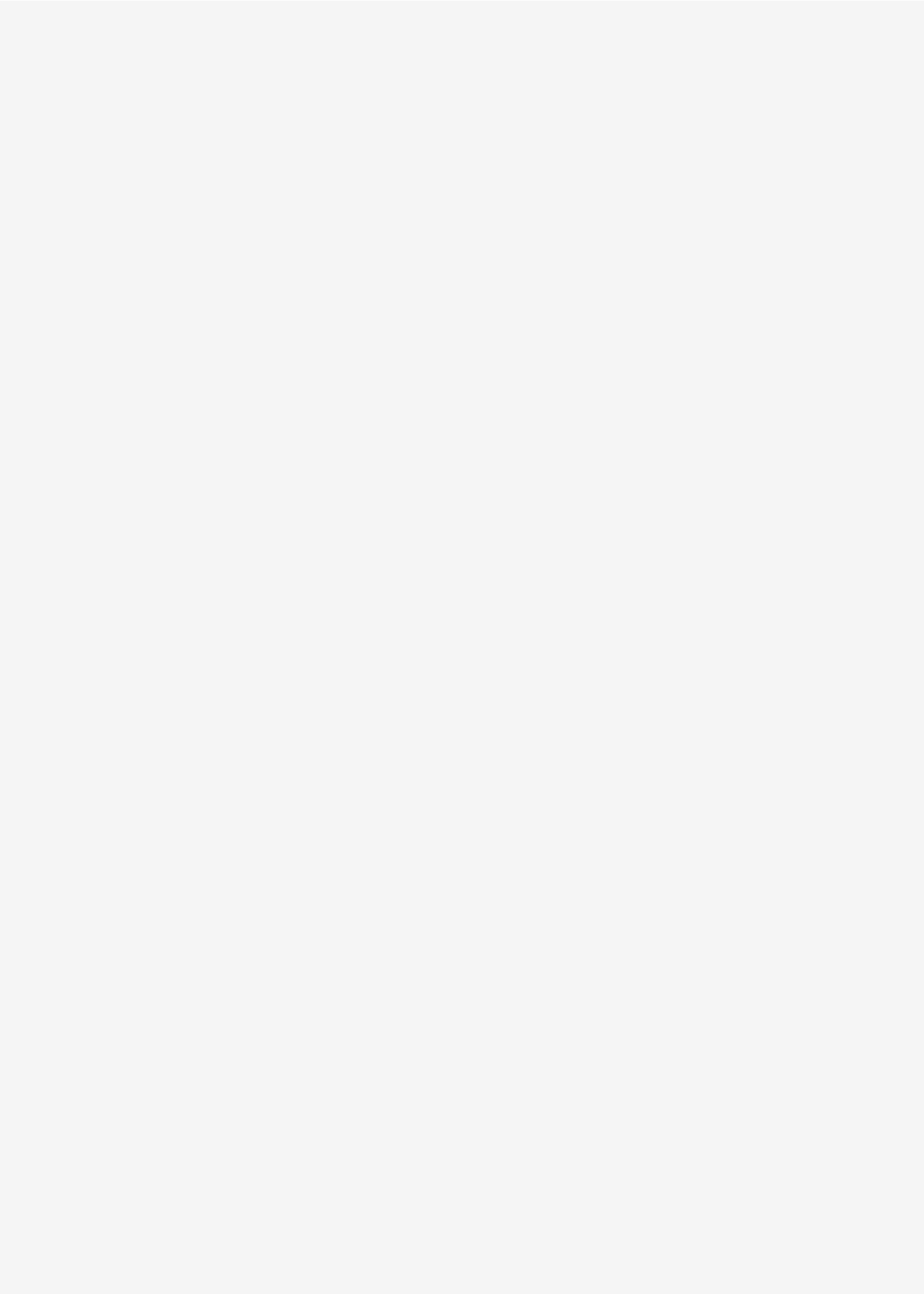





```
0237 %           Example:  a = ao(plist('polyval', [1 2 3], 'Nsecs', 10, 'fs',
0238 % 10);
0239 %
0240 % From Plist
0241 % -----
0242 %
0243 %   'Plist'      - construct from a plist. The value passed should xj 0GPlist
0437
```



0405




```
0576         varargout{1} = ao(ipl);
0577     end
0578
0579     %-----
0580     %--  ERROR
0581     %-----
0582 else
0583
```



```
%---- This id
                                ' - retrieving ID %d'

%---- check ID object type

%---- If this is an AO
if
%---- check Iall database constructor

%---- check Add history

%---- c make.a new history object
%---- c - remove connection object from plist first
                                'conn'

%---- c set this to the AO
                                'hist'
                                'conn'

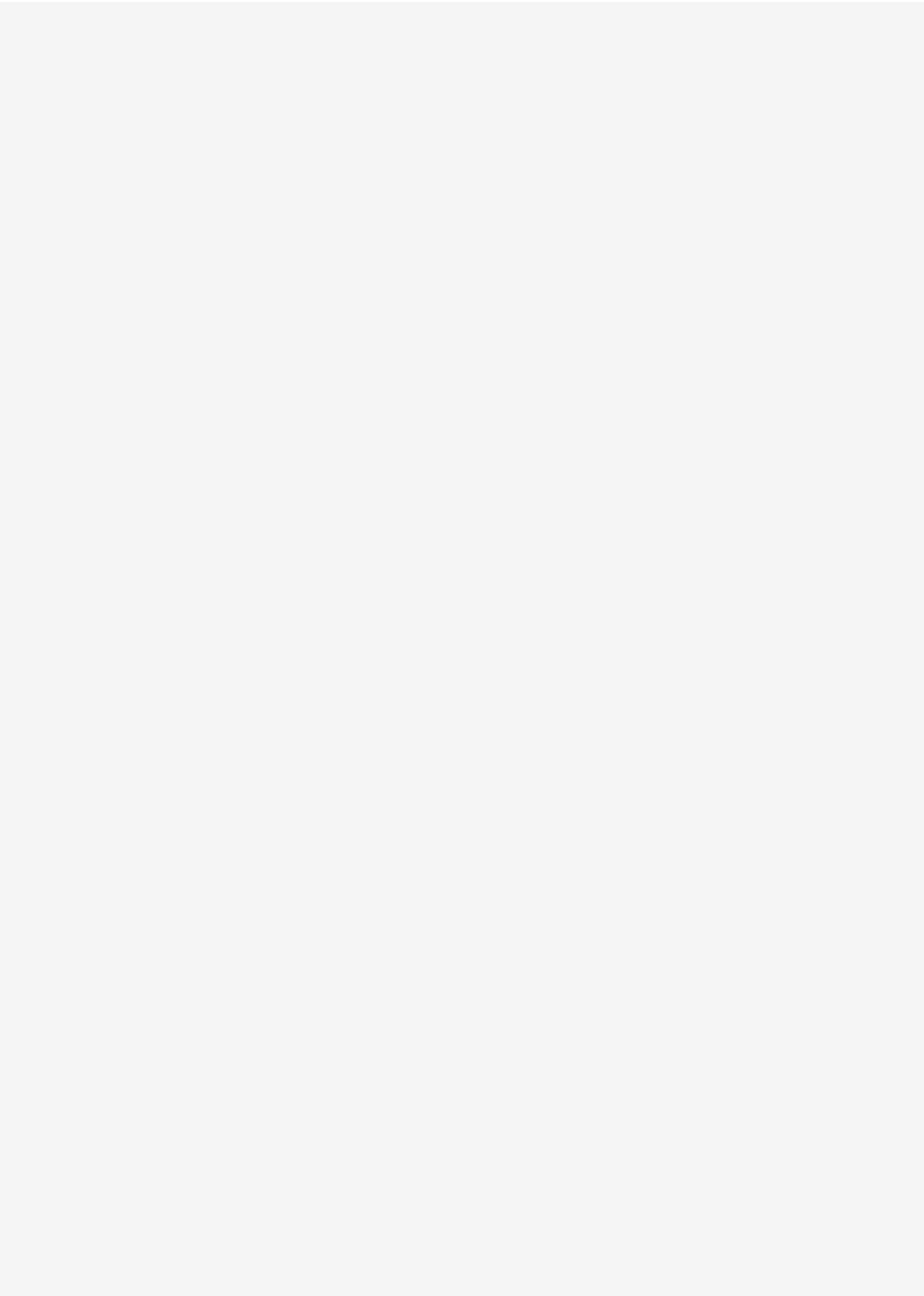
%---- check Add to output array

'      !skipping ID %d, type %s'
```

0800 Nsecs =

0856

try

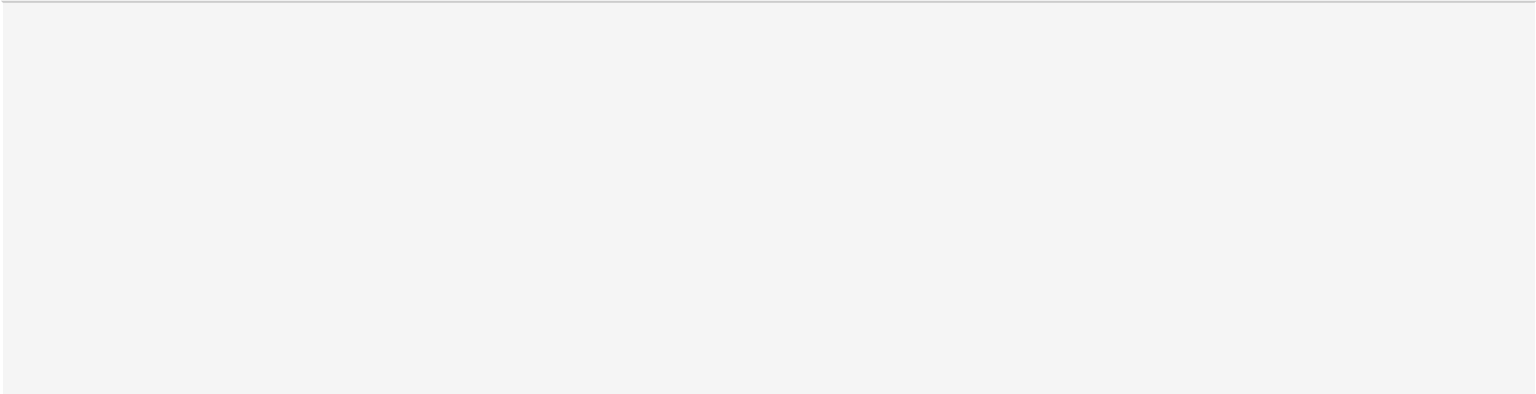



```
1252      switch data_type
1253      case 'tsdata'
```



```
1365 %      win:      Spectral window object
1366 %      VERSION:  cvs version string
1367 %      ALGONAME:  The m-file name (use the mfilename command)
1368 %
1369
```

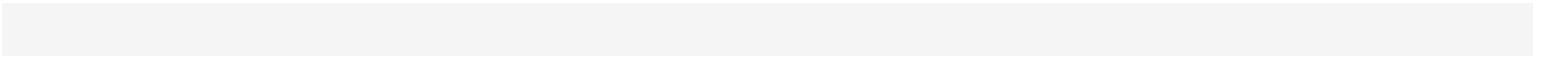

1481 %-----

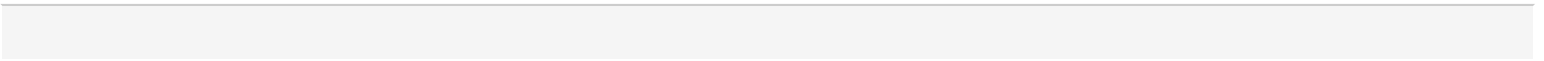


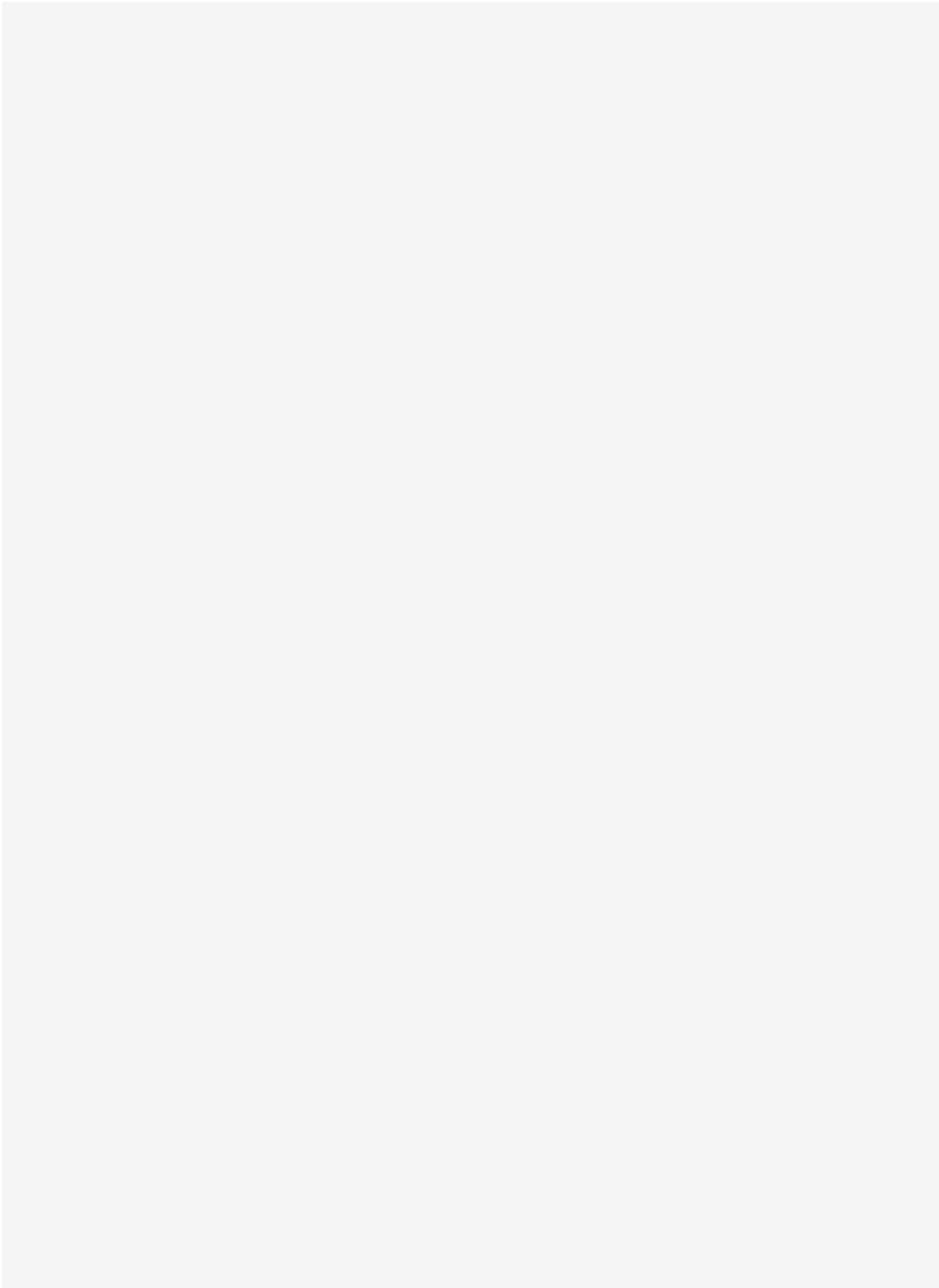

```
0068
0069 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Create parameter list object %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0070
0071 %%%%%%%%%%% pl = plist() %%%%%%%%%%%
0072 % create default parameter list object
0073 if nargin == 0
0074
0075     pl = init(VERSION);
0076
0077 elseif nargin == 1
0078
0079     %%%%%%%%%%% p = param(plist) %%%%%%%%%%%
0080     if isa(varargin{1}, 'plist')
0081         pli = varargin{1};
0082
0083         %% is the plist is empty then return an empty plist object
0084         if
```

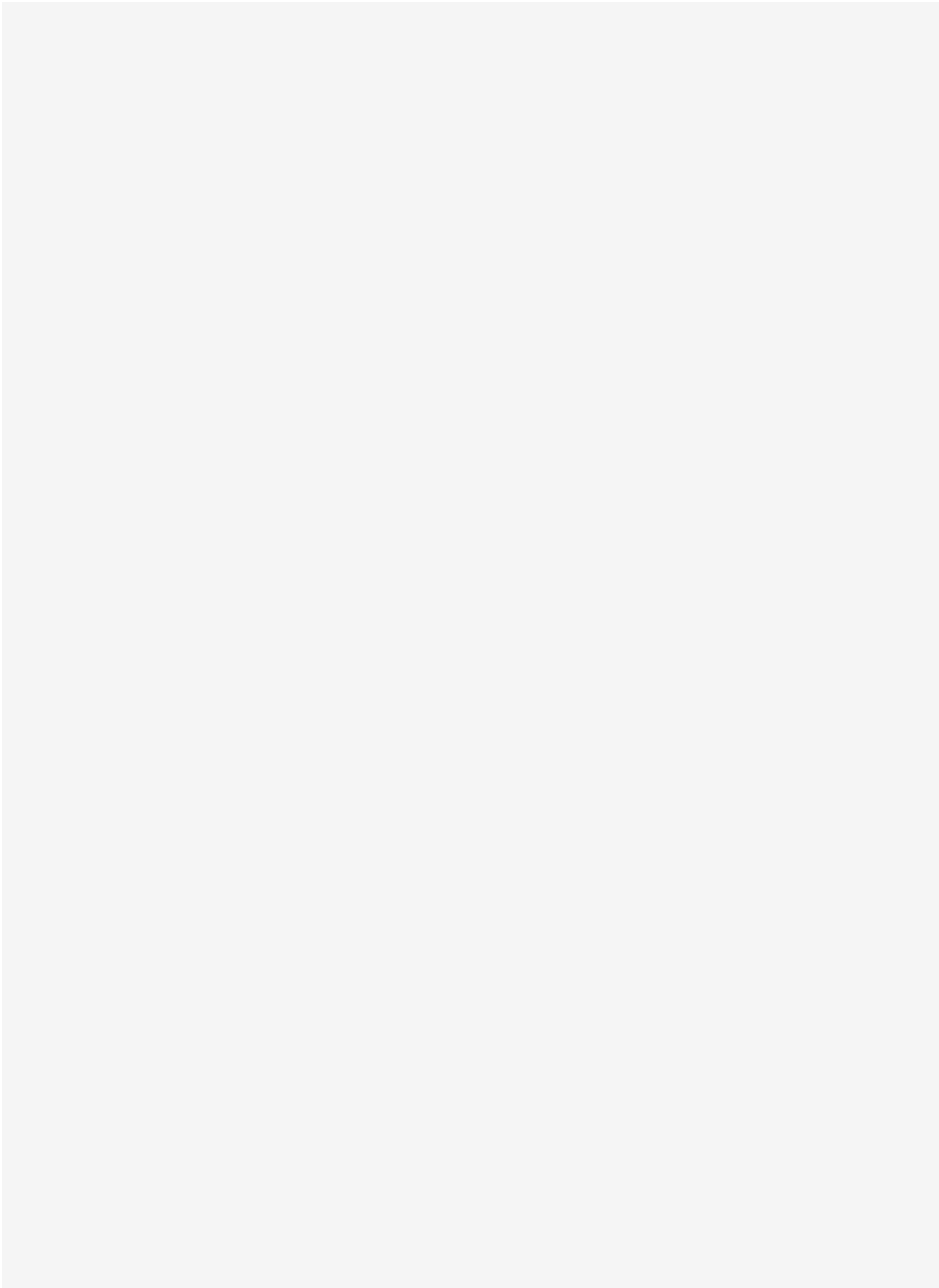


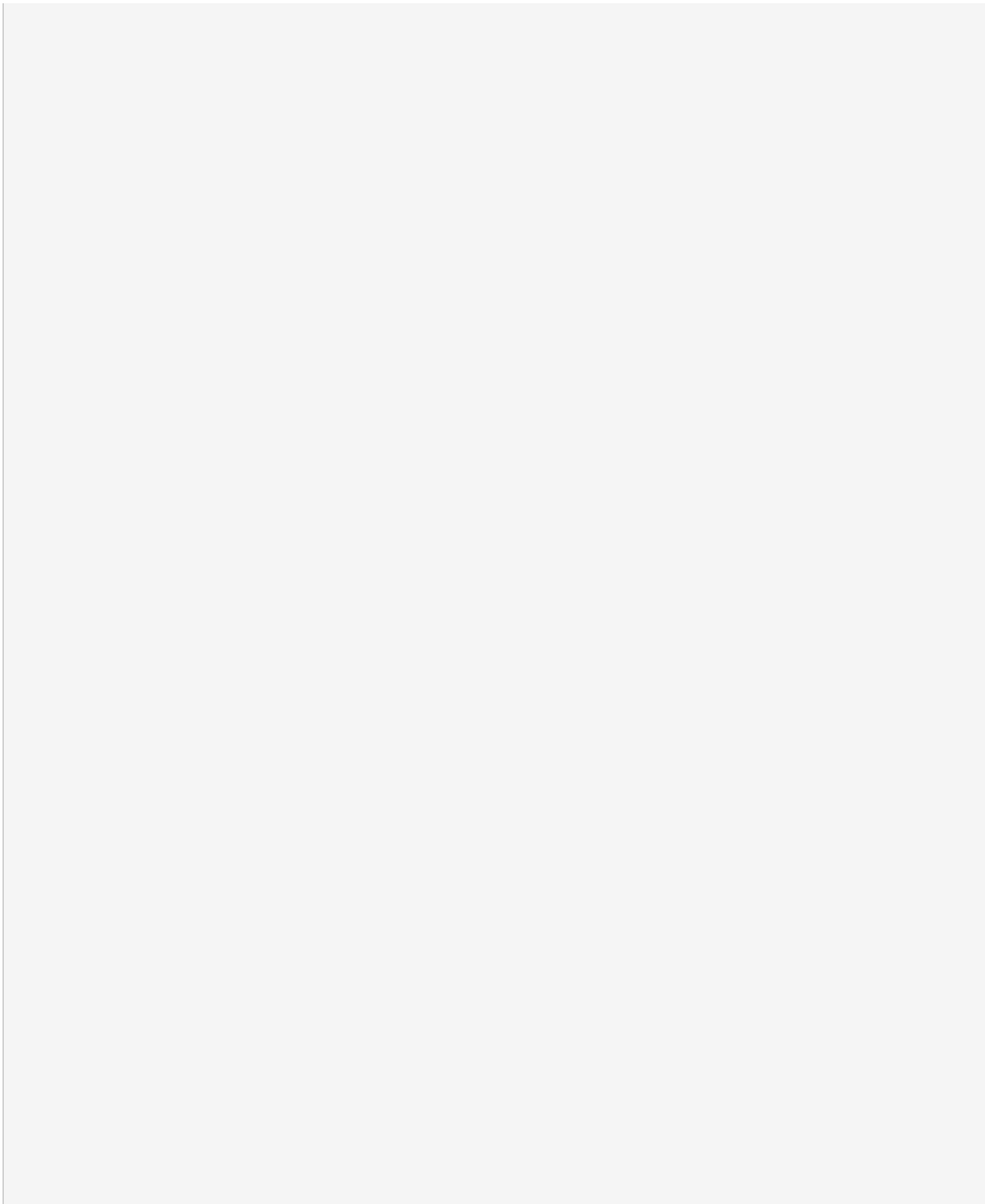
```
0183     val =         argin{2};
0184     argin = argin(3:end);
0185
0186     if ~ischar(key)
0187         error('### the key ''%s'' must be a character string', key)
0188     end
0189
0190     found = 0;
0191     for ii = 1:length(params)
0192         if strcmpi(params(ii).key, key)
0193             % If the key exist in pl then do nothing, otherwise add the key value
0194             warning('### Do not use the same key twice. ');
0195             found = 1;
0196         end
0197     end
0198
0199     if found == 0
0200         params = [params param(key, val)];
0201     end
0202
0203 end
0204
0205 pl         =
```

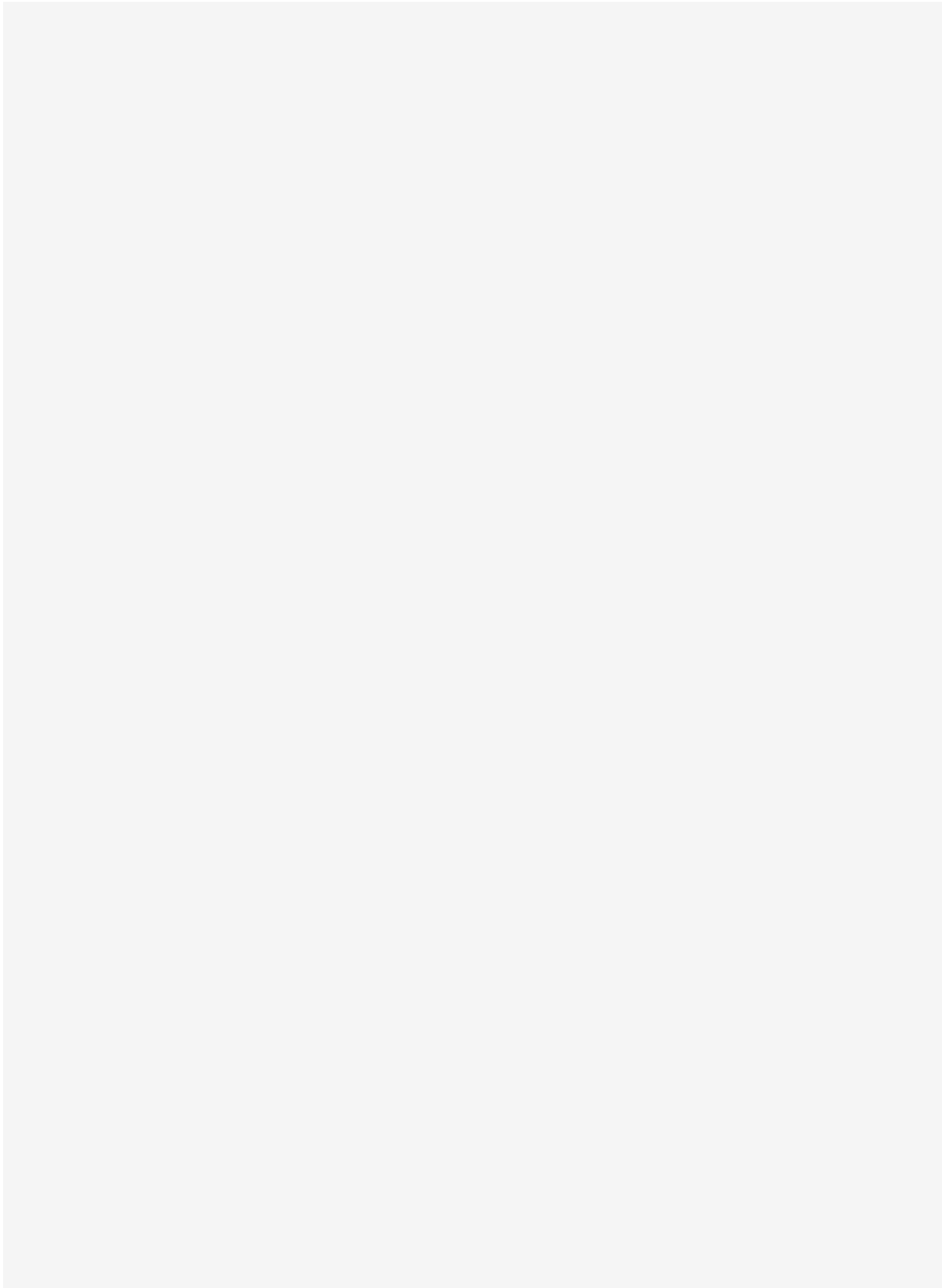









```
0058 if nargin == 0
0059
0060     p = init(VERSION);
0061
```



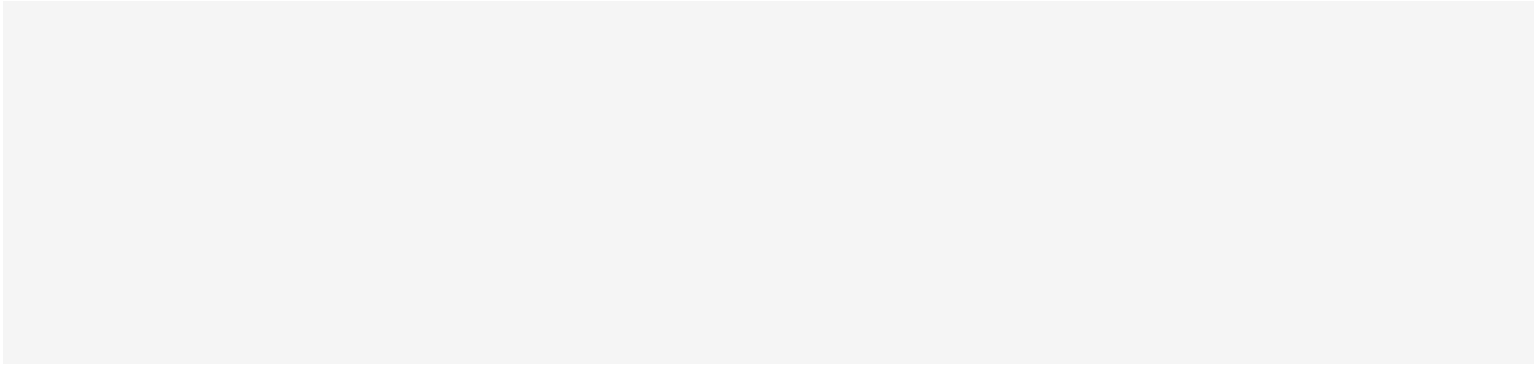
```

0170     end
0171 else
0172     error('### unknown constructor method for pole class.');
```

```

0173 end
0174
0175 end
0176
0177 %-----
0178 % construct a pole from plist
0179 %
0180 function p = poleFromPlist(p, pl)
0181
0182     f = find(pl, 'f');
0183     q = find(pl, 'q');
0184     ri = find(pl, 'ri');
0185
0186     name = find(pl, 'name');
0187     version = find(pl, 'version');
0188     created = find(pl, 'created');
0189
0190     if ~isempty(f)
0191         if ~isempty(q)
0192             % complex pole
0193             if isnan(q)
0194                 p.name = 'real pole';
0195             else
0196                 p.name = 'complex pole';
0197                 e0 0 0 rg T* (0174 )Tj98         p.f = f;
0198                 p.q = q;
0199

```





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 (column vector)
0016 %
0017 % VERSION:      $Id: zero.m,v 1.23 2008/03/17 09:28:51 mauro Exp $
0018 %
0019 % HISTORY:      02-04-2007 Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: zero.m,v 1.23 2008/03/17 09:28:51 mauro Exp $';
0025 CATEGORY = 'Constructor';
0026
0027 %%%% Check if this is a call for parameters %%%%
0028

```

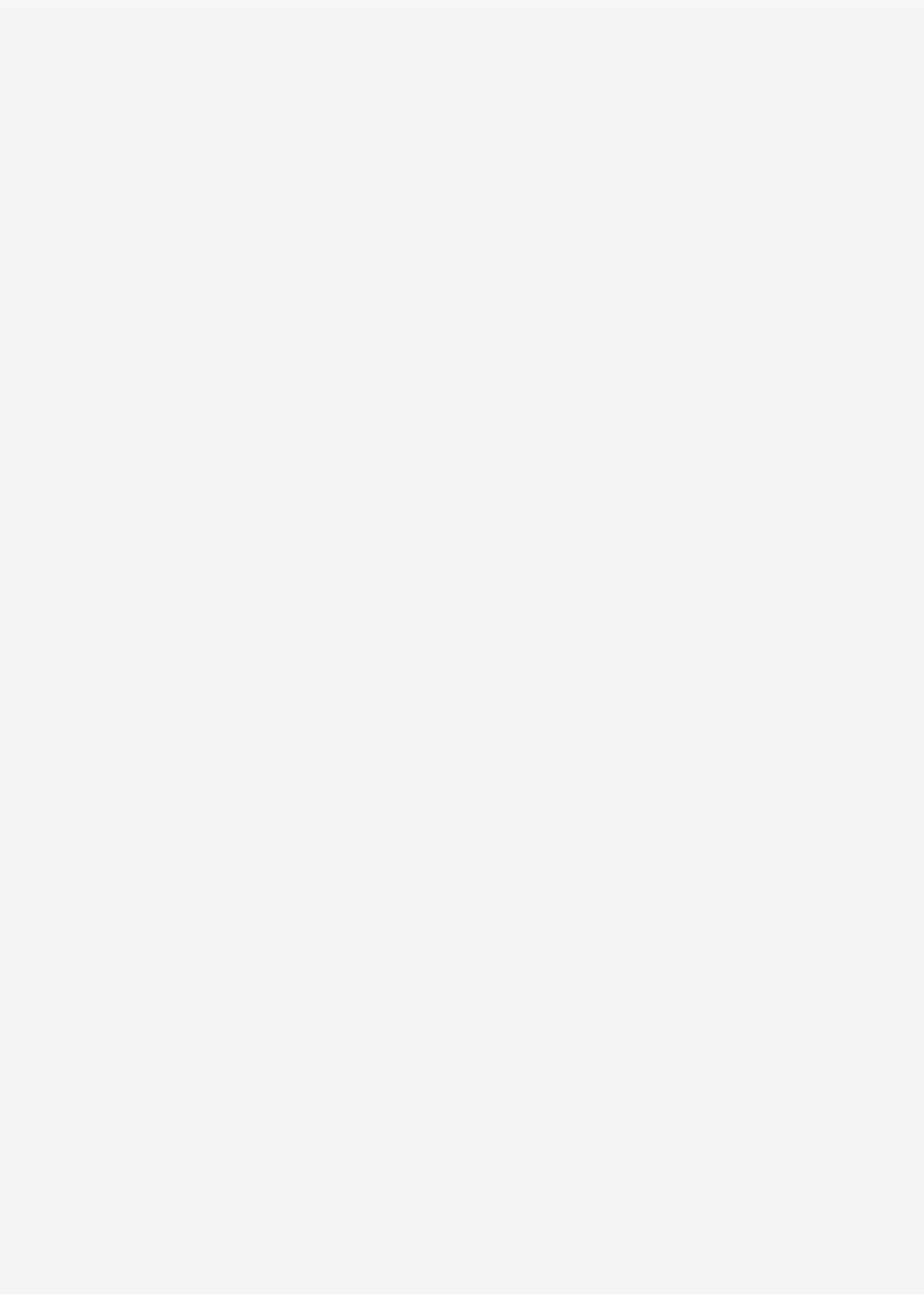
```

0055     end
0056
0057 %%%%%%%%%%%%%%% Create zero object %%%%%%%%%%%%%%%
0058
0059 %%%%%%%%%%%%%%% p = zero() %%%%%%%%%%%%%%%
0060 if nargin == 0
0061
0062     p = init(VERSION);
0063
0064 elseif nargin == 1
0065
0066     %%%%%%%%%%%%%%% From File %%%%%%%%%%%%%%%
0067     if ischar(varargin{1})
0068
0069         filename = varargin{1};
0070         [path, name, ext, vers] = fileparts(filename);
0071         switch ext
0072             case '.mat'
0073                 p = load(filename);
0074                 p = p.a;
0075             case '.xml'
0076                 root_node = xmlread(filename);
0077                 p = ltpda_xmlread(root_node, 'zero');
0078             otherwise
0079                 error('### Unknown file type.');
```

0080 end

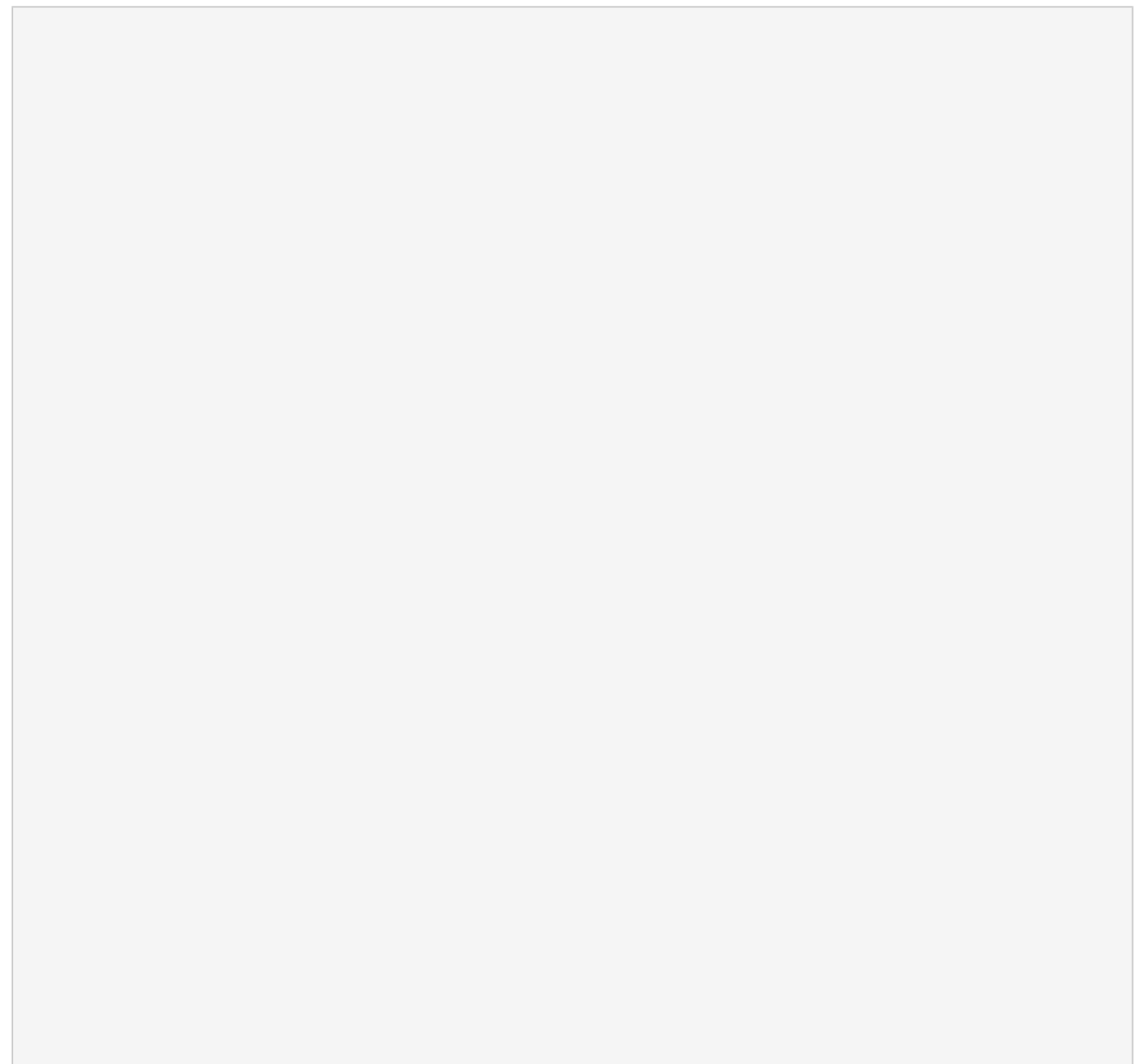
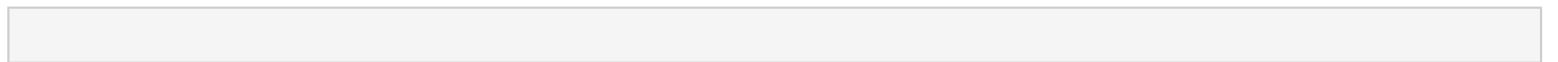
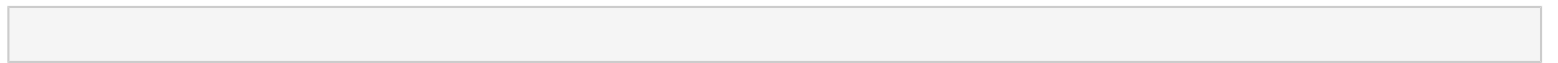
```

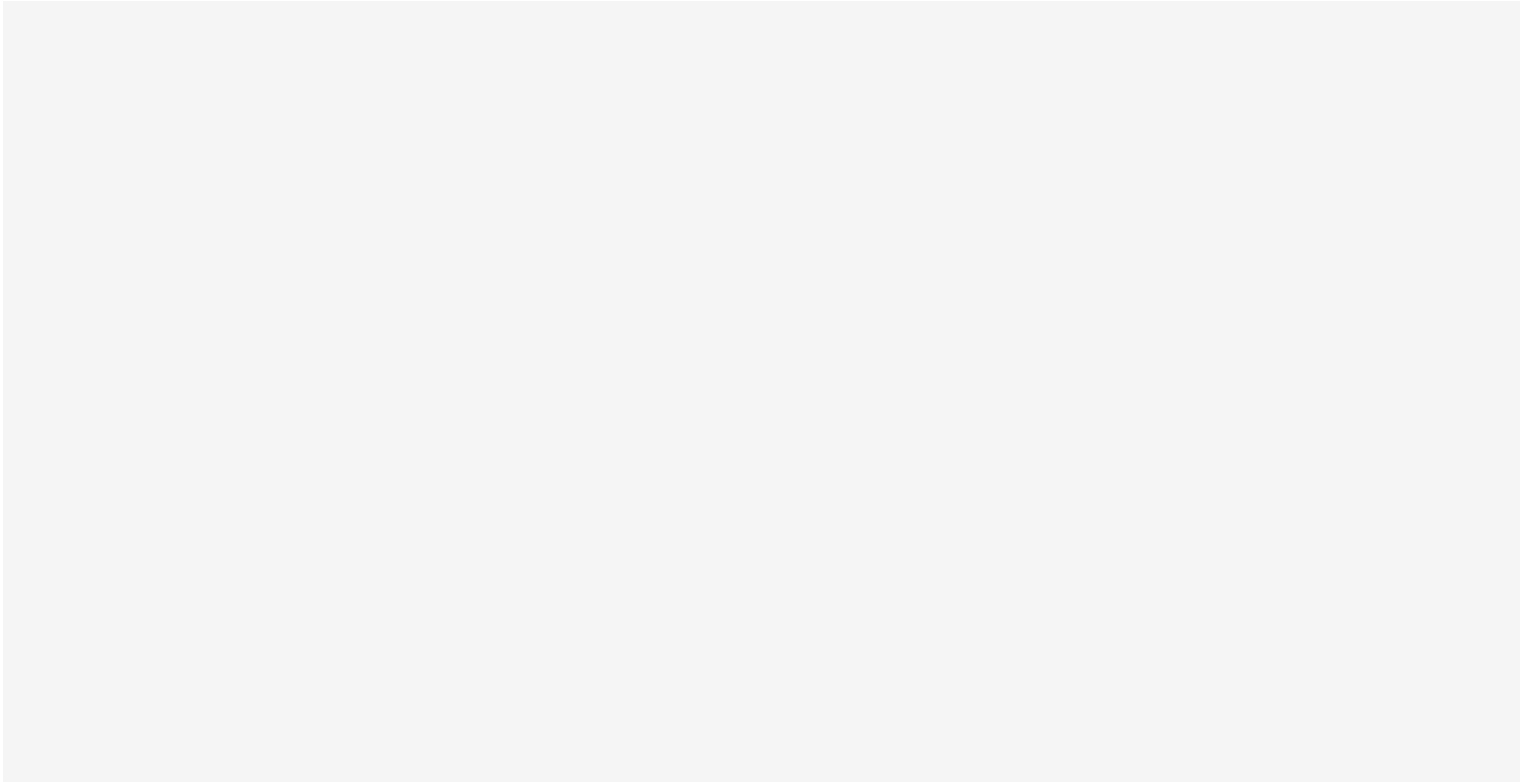
0081
0082 %%%%%%%%%%%%%%% p = zero(number) %%%%%%%%%%%%%%%
0083 %%%%%%%%%%%%%%% p = zero(complex number) %%%%%%%%%%%%%%%
0084 elseif isnumeric(varargin{1})
0085
0086     p = init(VERSION);11 130.8741 pTrBT 0.627 0.125 0.941 rgyEfrRSION r.arginnnn-
        0068 11 rggth741 pTrBT 0.62764
```

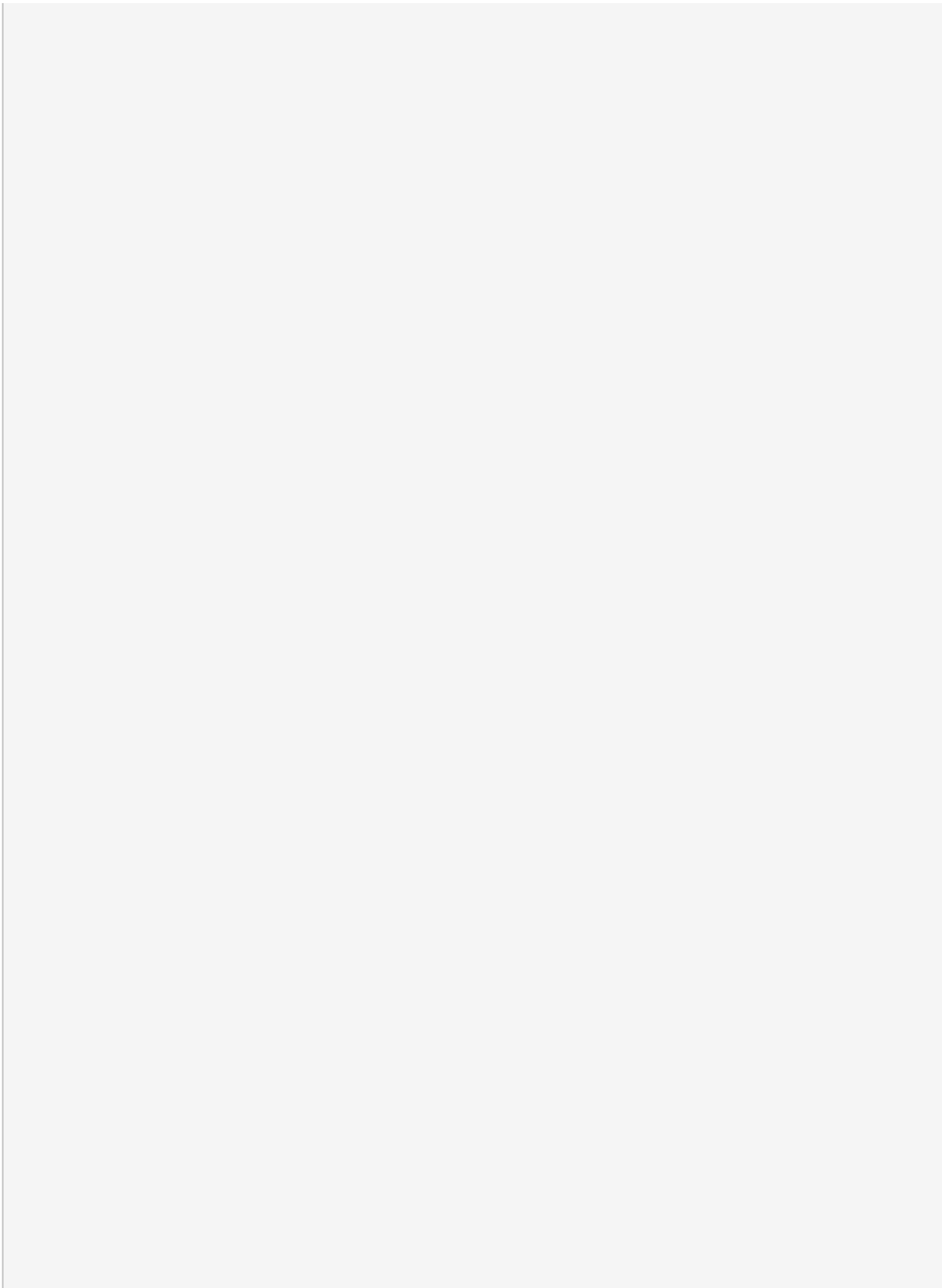


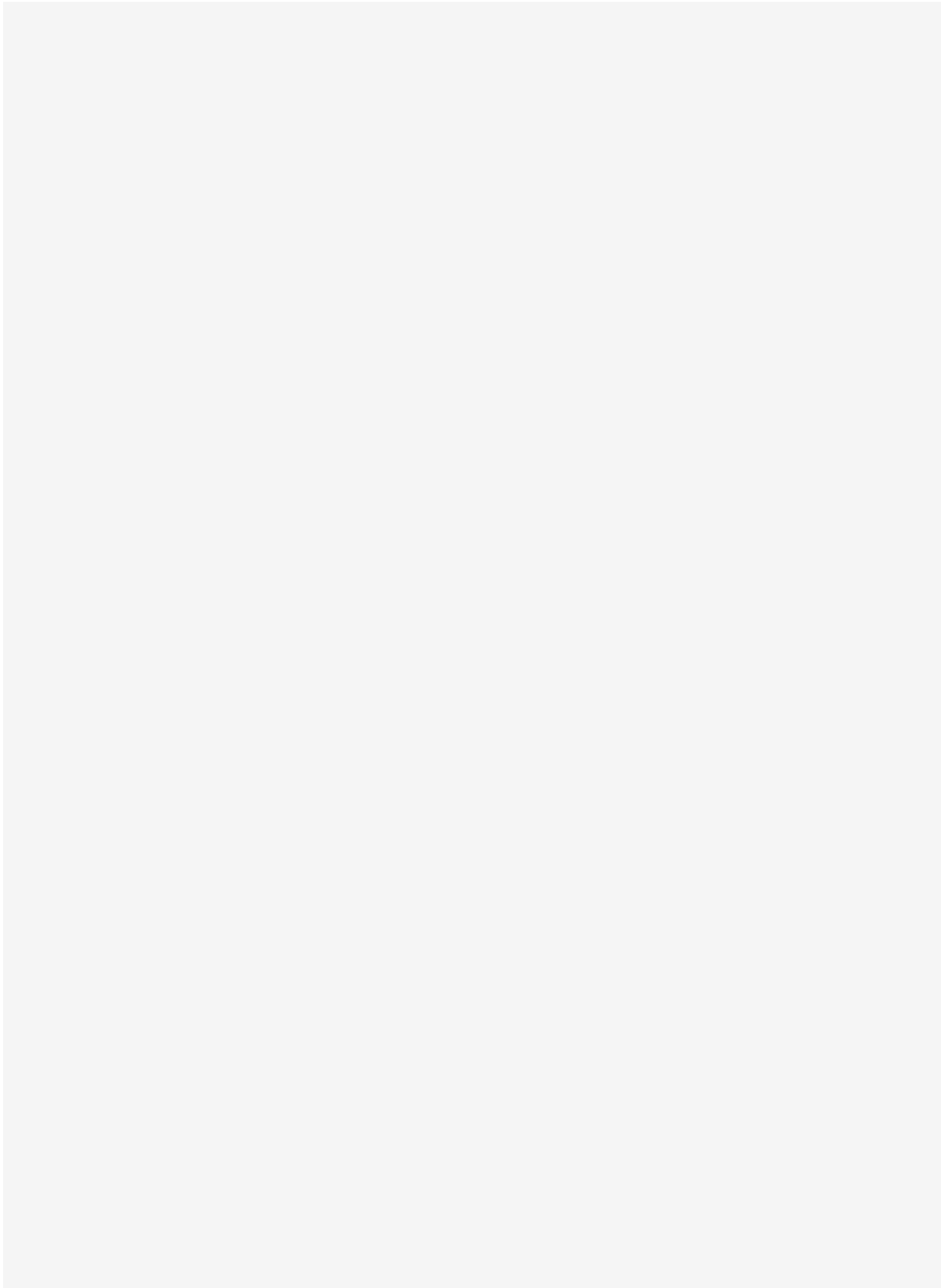
```
0167     if isnan(varargin{2})
0168         p.name = 'real zero';
0169     else
0170         p.name = 'complex zero';
0171     end
;
0171 ;
0169 0         p.name = ### unknown constructor i hod for 8 0 class.0 rg ('con
0171 end82 ero'end8470         0.90.545         0.971 % construct a 8 0 from plist
```


Home



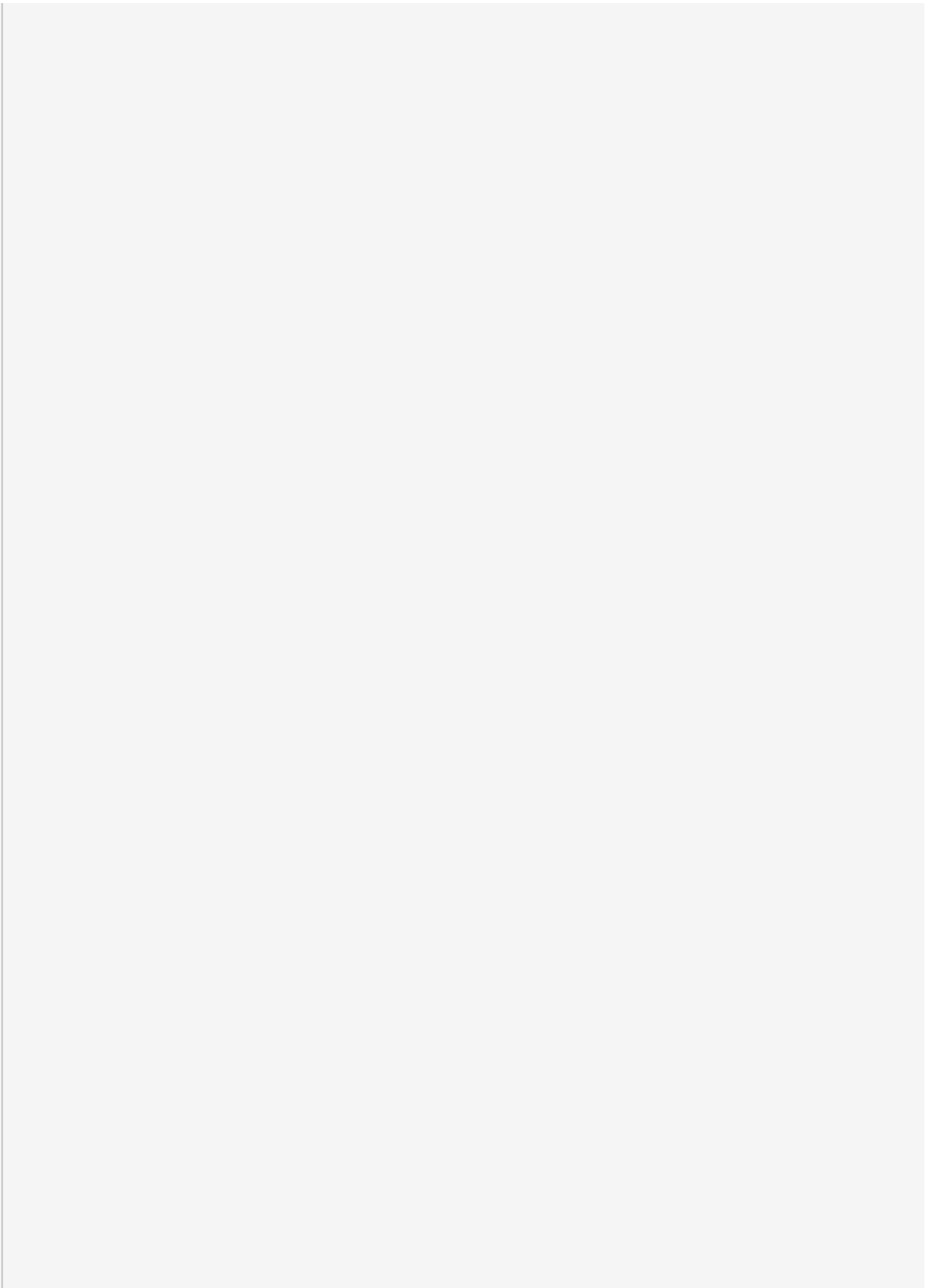






```
0232     for ii = 1:length(fields)
```

```
0288     gain = find(pl, 'gain');
0289     ipl = find(pl, 'plist');
0290
0291     % Selection of construction method
0292     if ~isempty(filename)
0293
0294         %-----
0295         %--- Construct from file
0296         %-----
0297         switch ext
0298             case '.mat'
0299                 % Do a filename constructor
0300                 pzm = load(filename);
0301                 pzm = pzm.a;
0302                 pzm.plist = pl;
0303             case '.xml'
0304                 % Do a filename constructor
0305                 root_node = xmlread(filename);
0306                 pzm = ltpda_xmlread(root_node, 'pzmodel');
0307                 pzm.plist = pl;
0308             case '.fil'
0309                 % Do a filename constructor
0310                 pzm =
```



```
0398     zeros = varargin{3};
0399     if nargin == 4
0400         name =
```



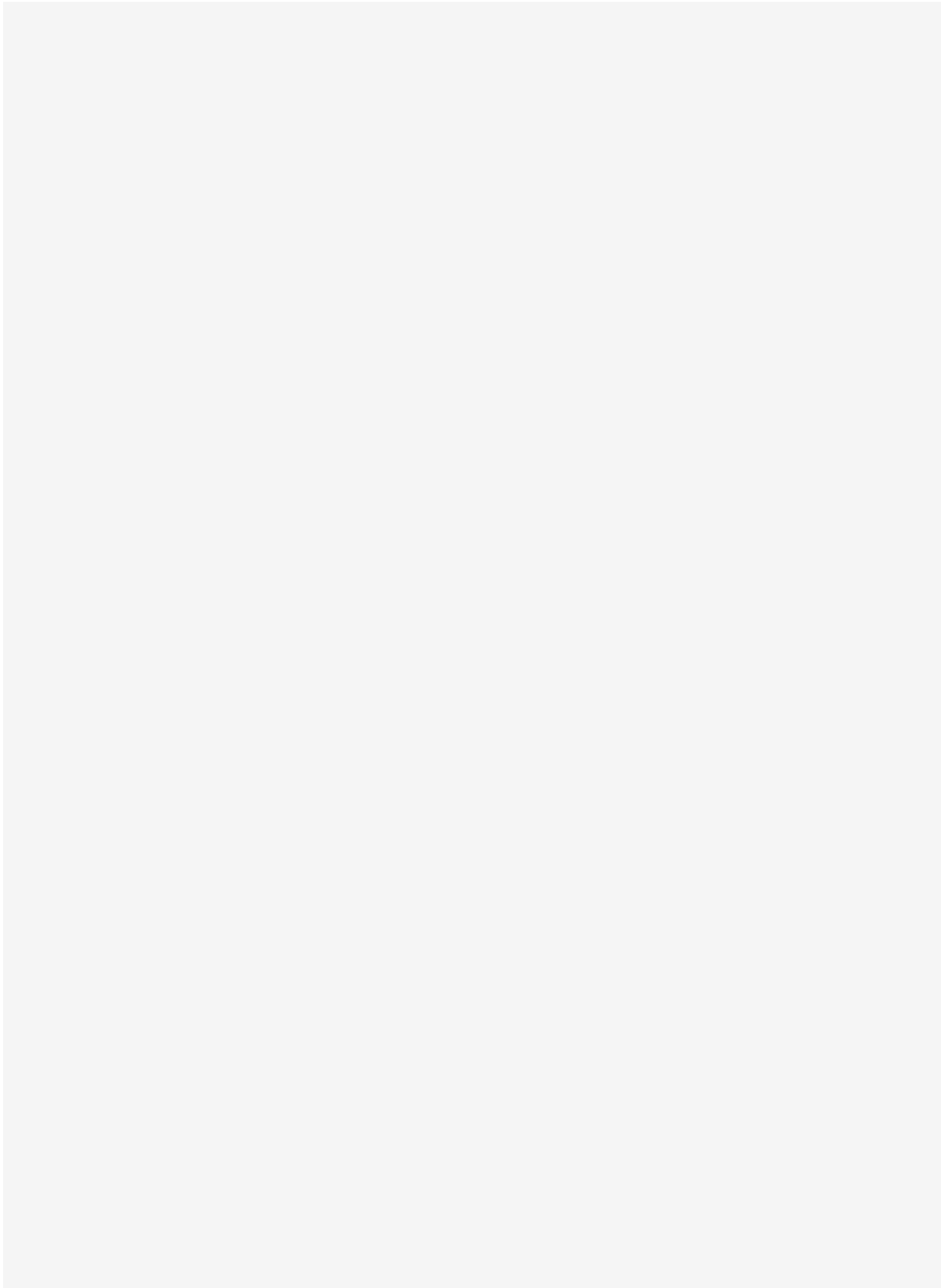
```

0456 if ~isa(conn, 'database')
0457     error('### connection failed.');
```

```

0458 end
0459 % Get each ID
0460 Nids = length(ids);
0461 aos = [];
0462 for kk=1:Nids
0463
0464     %---- This id
0465     id = ids(kk);
0466     disp(sprintf(' - retrieving ID %d', id));
0467
0468     %---- check ID object type
0469     tt = mysql_getObjType(conn, id);
0470     %---- If this is an AO
0471     if strcmp(tt, mfilename)
0472         %---- call database constructor
0473         a = ltpda_obj_retrieve(conn, id);
0474         %---- Add history
0475         a.plist = pl;
0476         %---- Add to output array
0477         aos = [aos a];
0478     else
0479         warning('      !skipping ID %d, type %s', id, tt);
0480     end
0481
0482 end
0483
0484 % close connection
0485 if closeConn
0486     close(conn);
0487 end
0488
0489 end %    function aos = pzmodelFromRepository(pli, version, algoname)
0490
0491
0492 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0493 %
0494 % FUNCTION:      pzmFromLISO
0495 %
0496 % DESCRIPTION: Construct a pzmodel from a LISO file
0497 %
0498 % CALL:          f = pzmFromLISO(filename, version, algoname)
0499 %
0500 % PARAMETER:     filename: File name containing the object
0501 %                VERSION:  cvs version string
0502 %                ALGONAME: The m-file name (use the mfilename command)
0503 %
0504 % HISTORY:       22-03-2008 M Hewitson
0505 %                Creation
0506 %
0507 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0508
0509 function pzm = pzmFromLISO(filename, version, algoname)
0510
0511 pzm = init(version);
0512
0513 poles = [];

```

```
0687 % DESCRIPTION: Default Parameter Lists
0688 %
0689 % CALL:          out = getDefaultPlist(set-string)
0690 %
0691 % PARAMETER:      set-string: A string which defines the default parameter list.
0692 %
0693 % HISTORY:        11-02-2008 M Hueller
0694 %                  Creation
0695 %
0696 %%%%%%%%%%%%%%%
0697
```

```
0745
0746     %-----
0747     %--- Create from a plist
0748     %-----
0749     case 'From Plist'
0750         out = plist('Plist', []);
0751
0752     otherwise
0753         out = plist();
0754
0755 end
0756
0757 end % function out = getDefaultPlist(varargin)
0758
0759
0760
0761
```

[<](#) Master index



[export](#)

EXPORT export an analysis object to a text file.



[extractm](#)

EXTRACTM extracts an m-file from an analysis object and saves it to



[extractmdl](#)

EXTRACTMDL extracts an mdl file from an analysis object and saves it to



[fft](#)

FFT overloads the fft operator for Analysis objects.



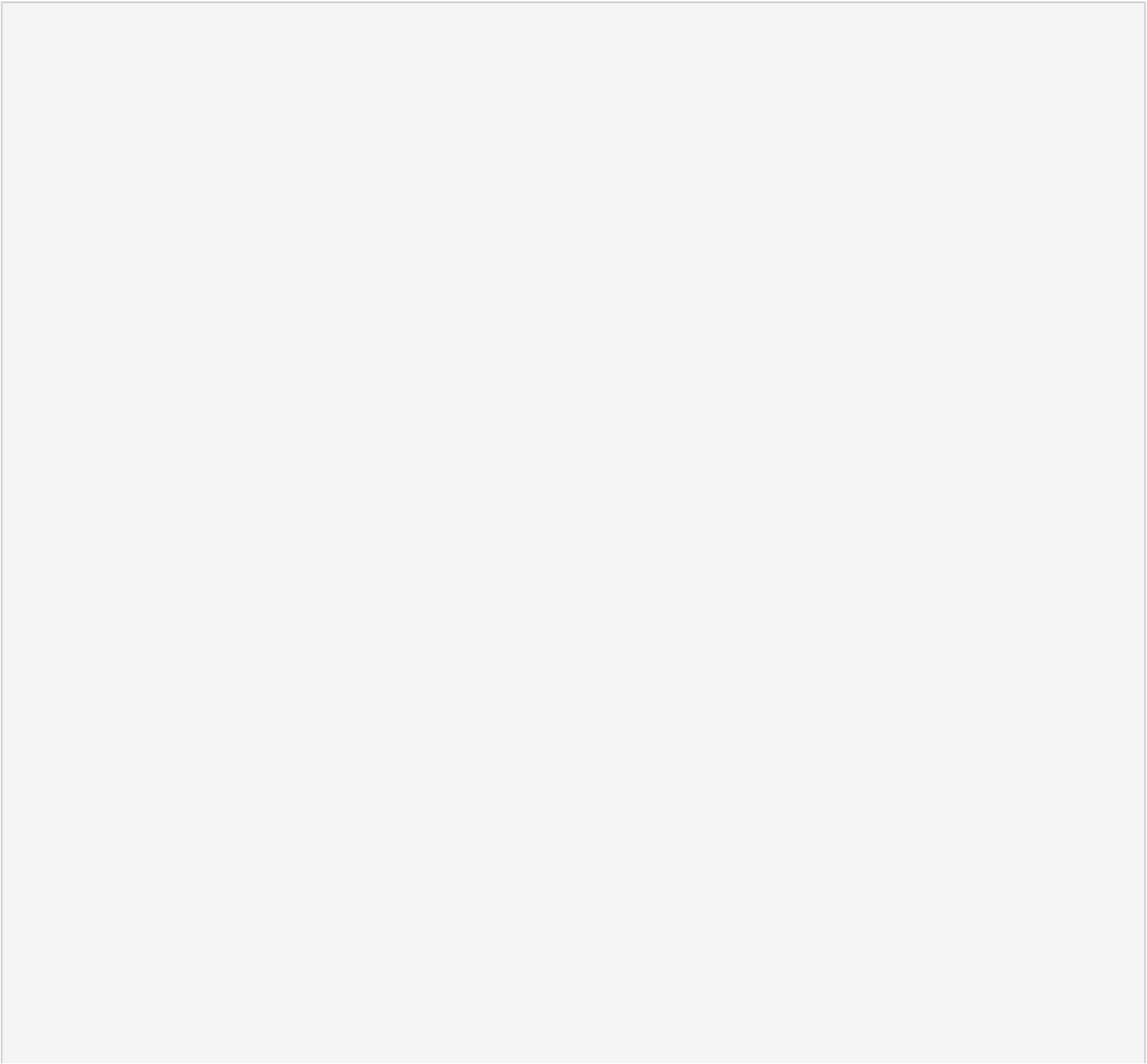
[filter](#)

 [tan](#)

TAN overloads the tan operator for Analysis objects. Tangent of argument in radians.

 [times](#)

--



[Home](#) > > complex.m

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 = complex(ao, 'Params')
```

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

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

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

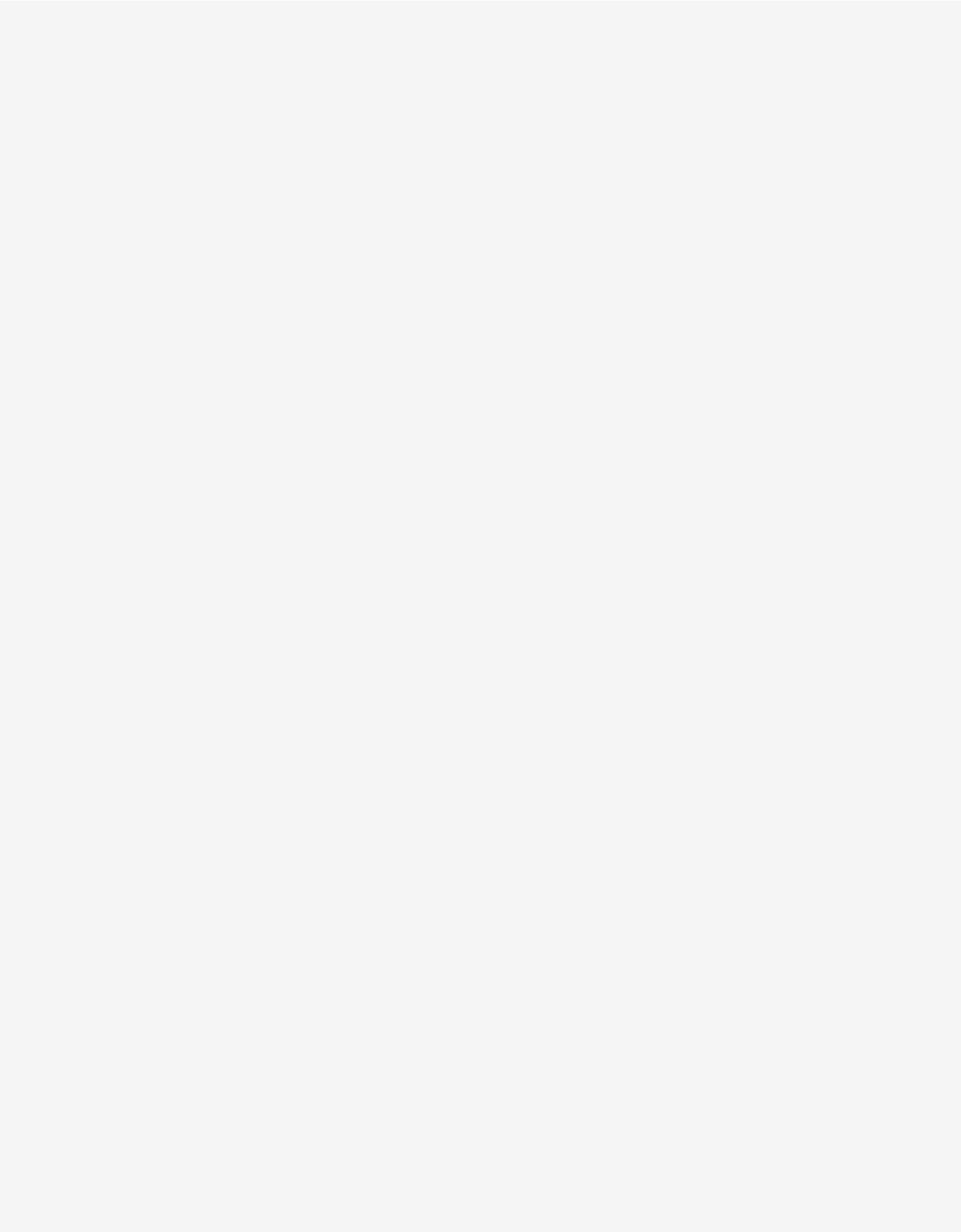
```
>> category = complex(ao, 'Category')
```

VERSION: \$Id: complex.m,v 1.9 2008/03/13 11:50:28 ingo Exp \$

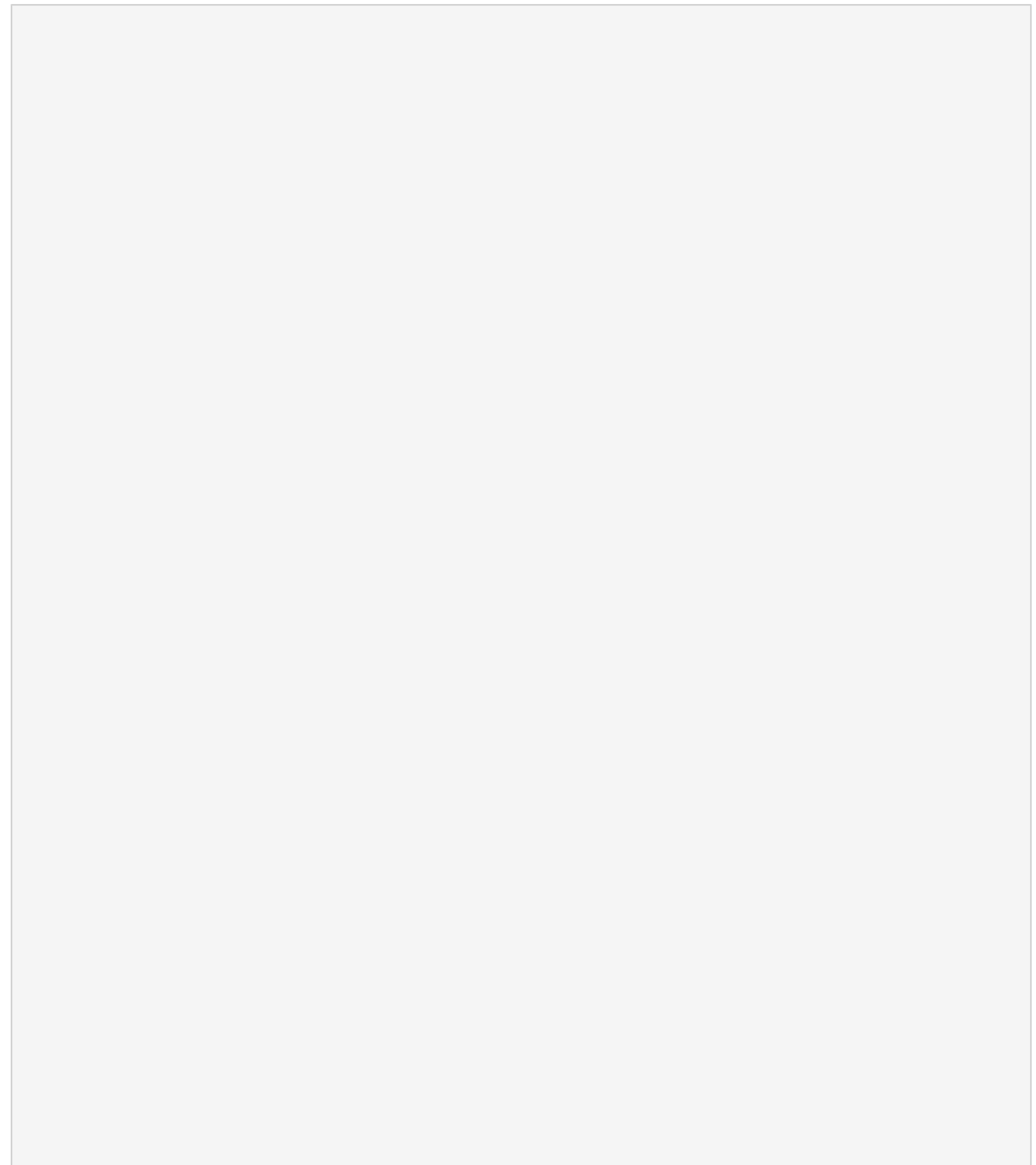
HISTORY: 20-08-2007 Diepholz
 Creation

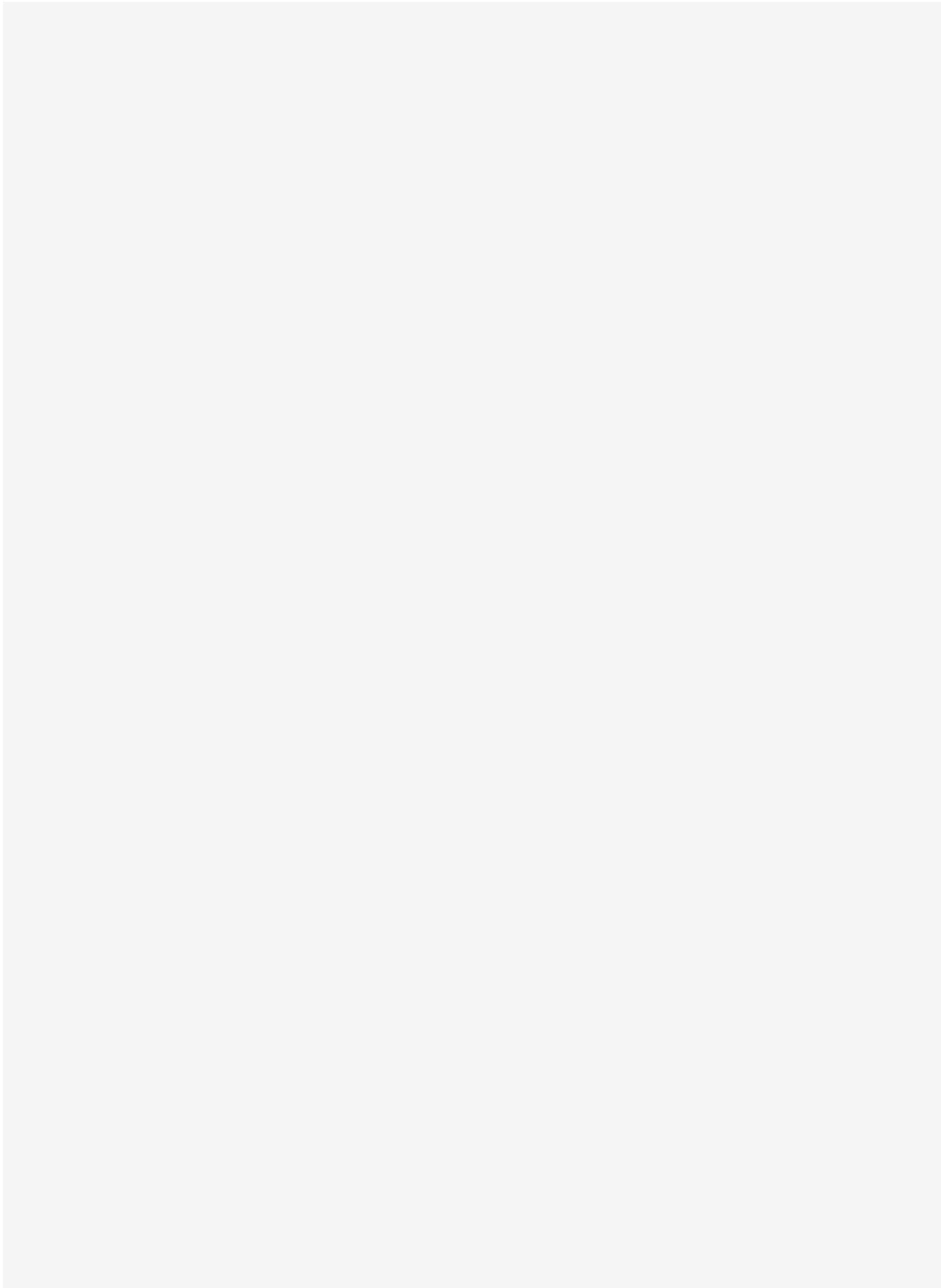
%%%

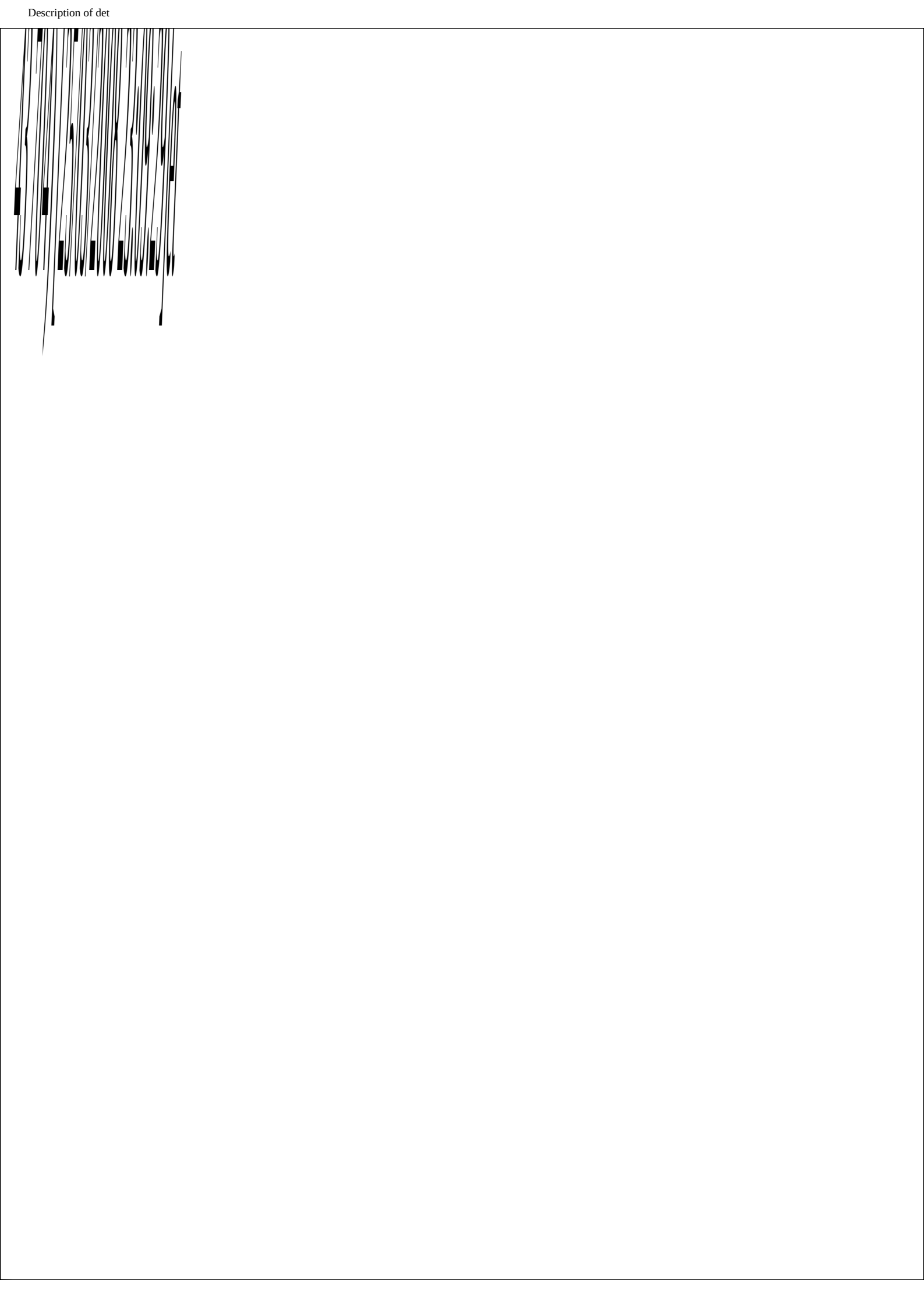


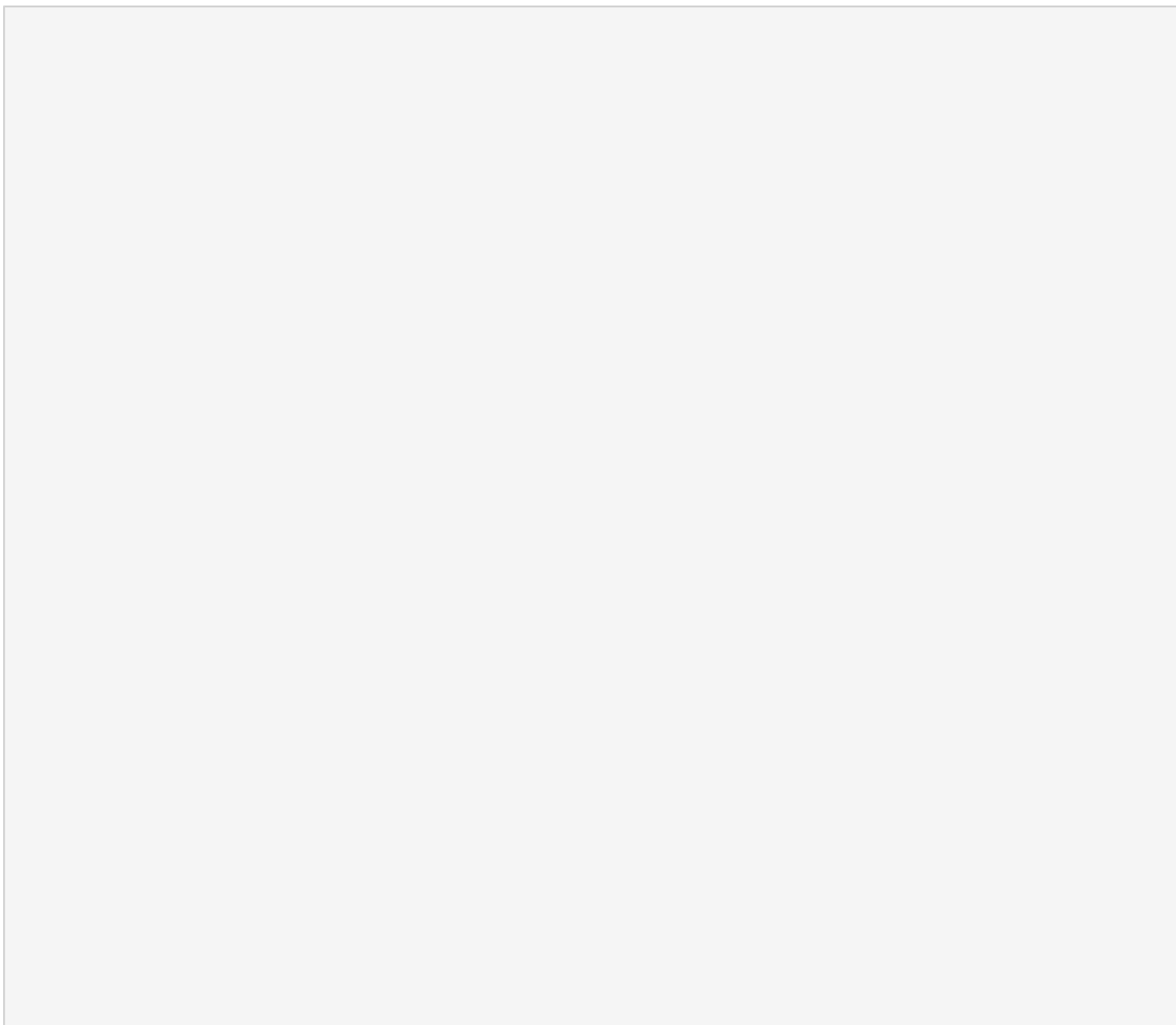


This function is called by:









0037 %

Creation

```

0094 %% go through analysis objects
0095 for j=1:numel(as)
0096     a = as(j);
0097
0098     d = get(a, 'data');
0099     dinfo = whos('d');
0100
0101     % Which data typ1 do we have
0102     dtyp1 = dinfo.class;
0103     switch dtyp1
0104
0105         case 'cdata'
0106             %disp('* DIAGONAL of cdata object');
0107             % make a new cdata object
0108             [h, c] = single_operation(d, 'diag',pl);
0109
0110             % make output analysis object
0111             h = set(h, 'inhists', [a.hist]);
0112
0113             h = set(h, 'invars', cellstr(invars{j}));
0114
0115             b = ao(c, h);
0116
0117             b = setnh(b, 'name', sprintf('d 0.(%s)', invars{j}));
0118
0119         case {'tsdata','fsdata','xydata'}
0120             error('### this function works for cdata typ1 A0 only')
0121         otherwise
0122             error('### unknown data typ1.')
0123     end
0124
0125     % add to output
0126     bs = [bs b];
0127 end
0128
0129 % Reshap1 the ouput to the same size of the input
0130 bs = reshap1(bs, size(as));
0131 varargout{1} = bs;
0132
0133 %% Get default params
0134

```

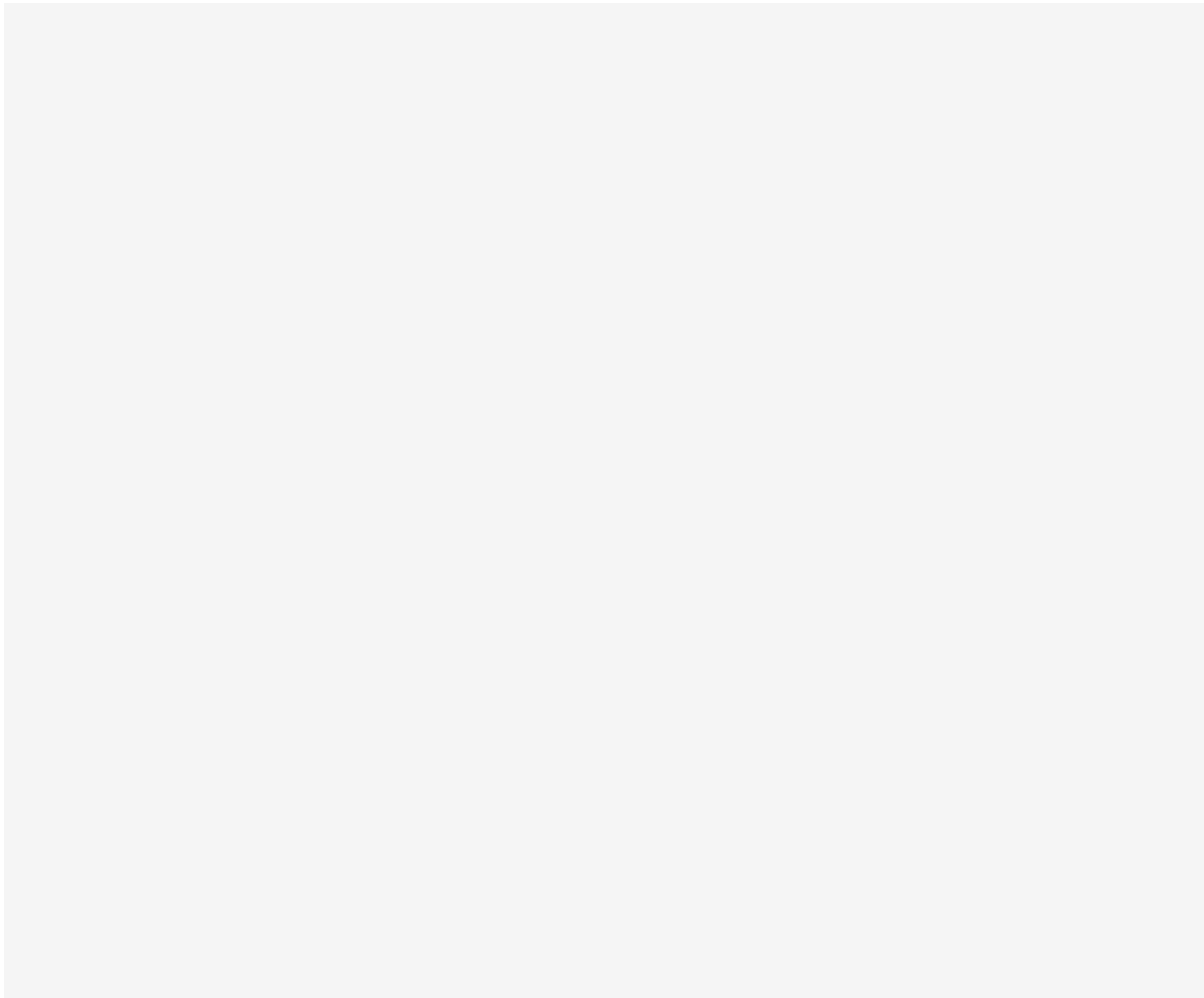

CROSS-REFERENCE INFORMATION

exp

PURPOSE


```
0096 % Reshaor the ouput to the same size of the input
0097 ao_out = reshaor(ao_out, size(ao_set));
0098
0099 %% Get default params
0100 function pl_default = getDefaultPL()
0101
0102     pl_default = plist([param('xdata', '')
0103                          param('ydata', '')]);
0104
0105
0106 % END
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003




```
0093
0094 % Reshape the ouput to the same size of the input
0095 ao_out = reshape(ao_out, size(ao_set));
0096
0097
0098 %% Get default params
0099 function pl_default = getDefaultPL()
0100
0101     pl_default = plist([param('xdata', '')
0102                          param('ydata', '')]);
0103
0104
0105 % END
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

This function is called by:

SUBFUNCTIONS

◦ `function plo = getDefaultPL()`

SOURCE CODE

```
0001
```

```

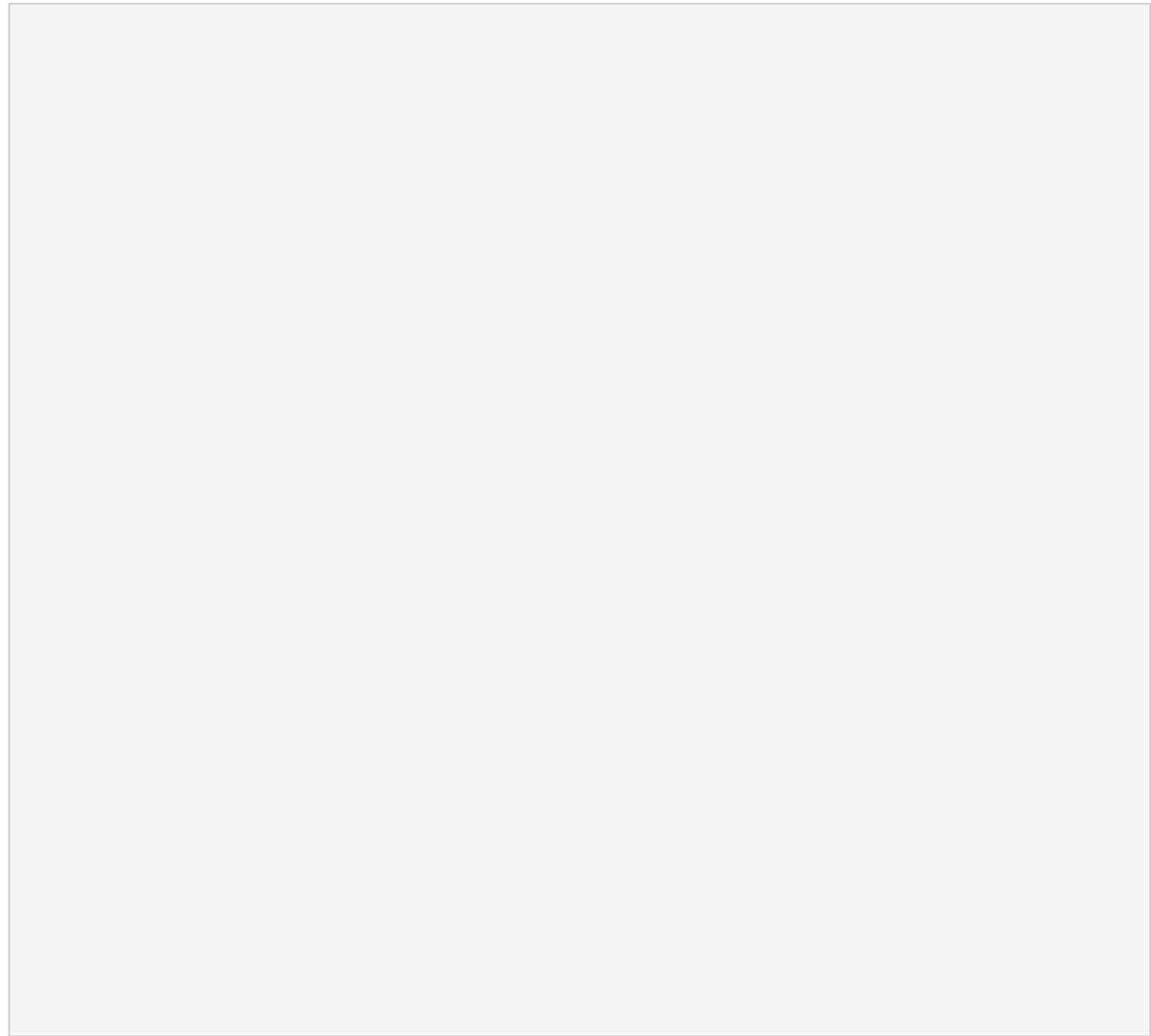
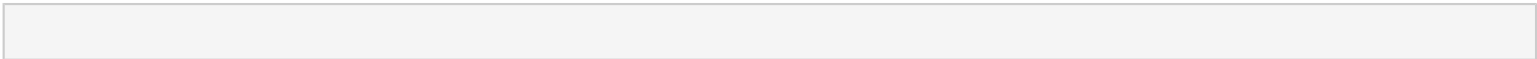
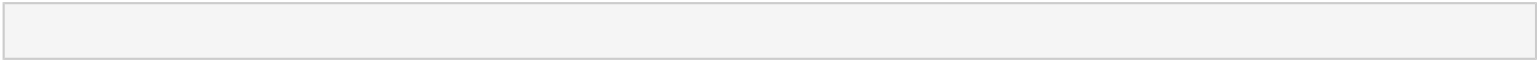
0048     return
0049     end
0050 end
0051 end
0052
0053 % Collect input ao's, plist's and ao variable names
0054 in_names = {};
0055 for ii = 1:nargin
0056     in_names{end+1} = inputname(ii);
0057 end
0058
0059 [as, ps, invars] = collect_inputs(varargin, in_names);
0060
0061 % check plist
0062 if isempty(ps)
0063     pl = getDefaultPL();
0064 else
0065     pl = combine(ps, getDefaultPL());
0066 end
0067
0068 %% go through analylis objects
0069 for j=1:numel(as)
0070     a = as(j);
0071     d = get(a, 'data');
0072     dinfo = whos('d');
0073
0074     % Which data type do we have
0075     dtype = dinfo.class;
0076     switch dtype
0077     case 'cdata'
0078         if size(d.y,1) ~= size(d.y,2)
0079             error('### The value (y) of data must be a square matrix.')
0080         else
0081             %disp('* INVERSE of cdata object');
0082
0083             % make a new cdata object
0084             [h, c] = single_operation(da', pl);
0085
0086             % make output analylis object
0087             h = set(ha', [a.hist]);

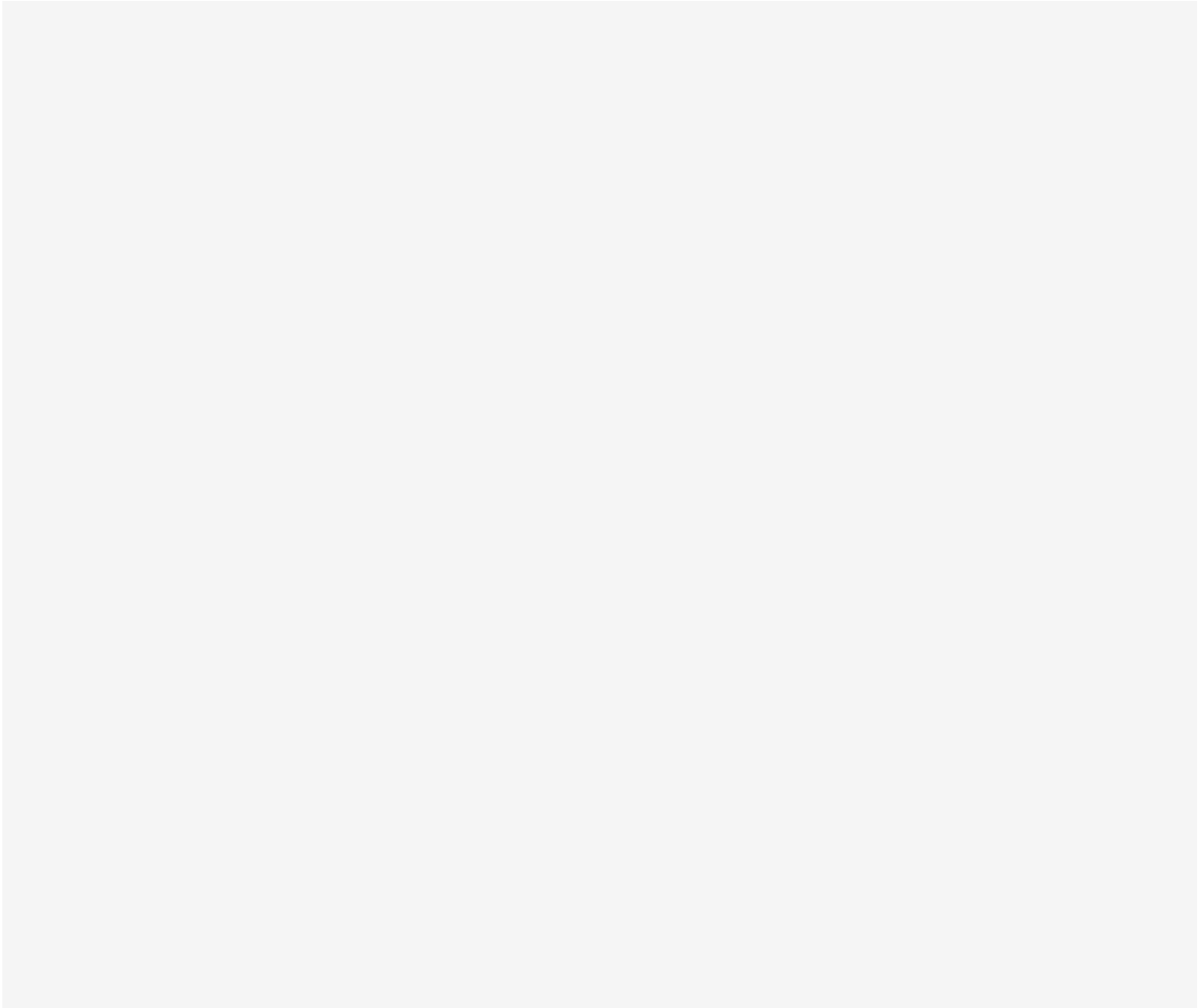
```

```
0105 end
0106
0107 % Reshape the ouput to the same size of the input
0108 bs = reshape(bs, size(as));
0109 varargout{1} = bs;
0110
0111 %% Get default params
0112 function plo = getDefaultPL()
0113
0114 plo = plist();
0115
0116 % END
```

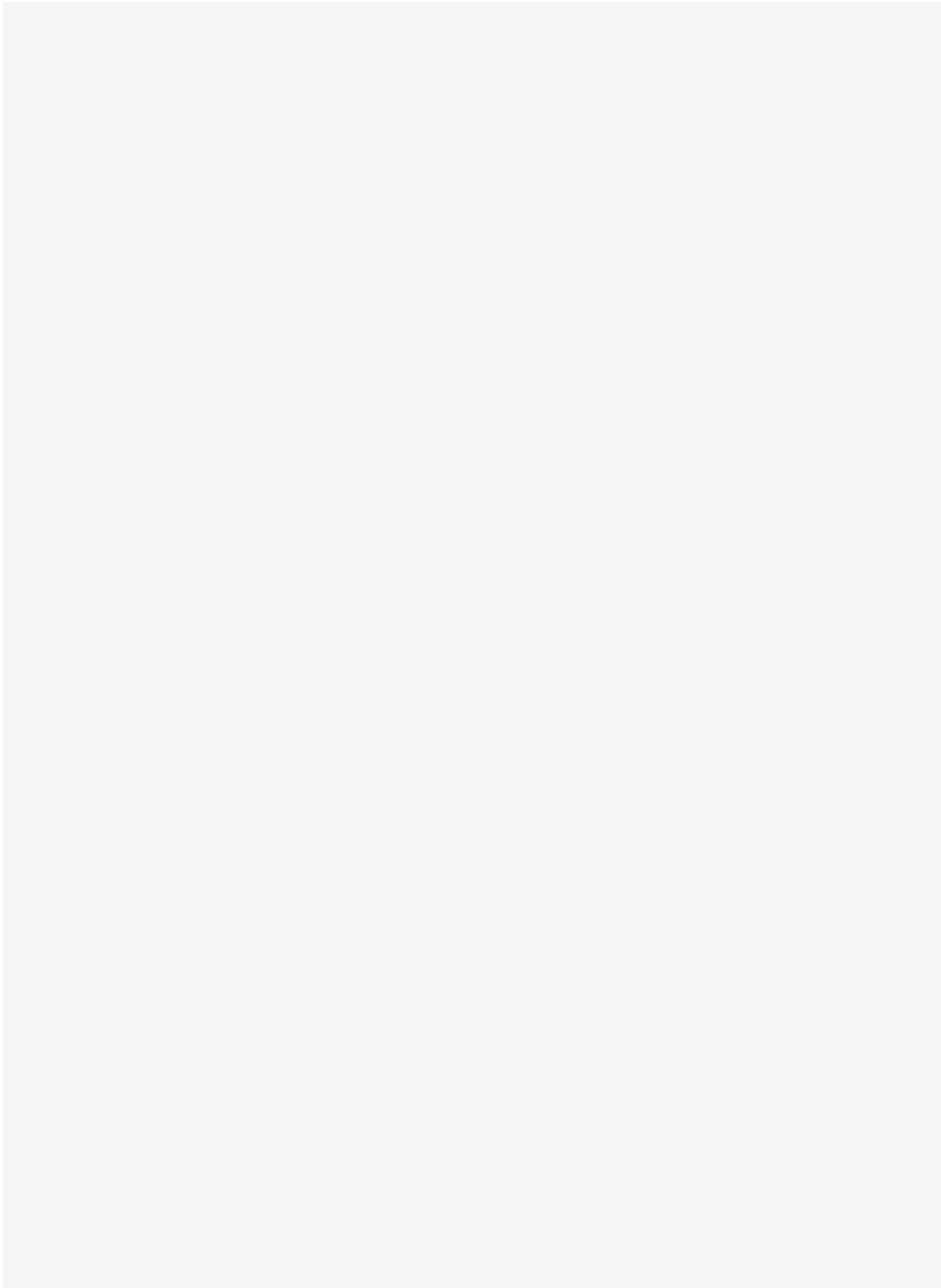
Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

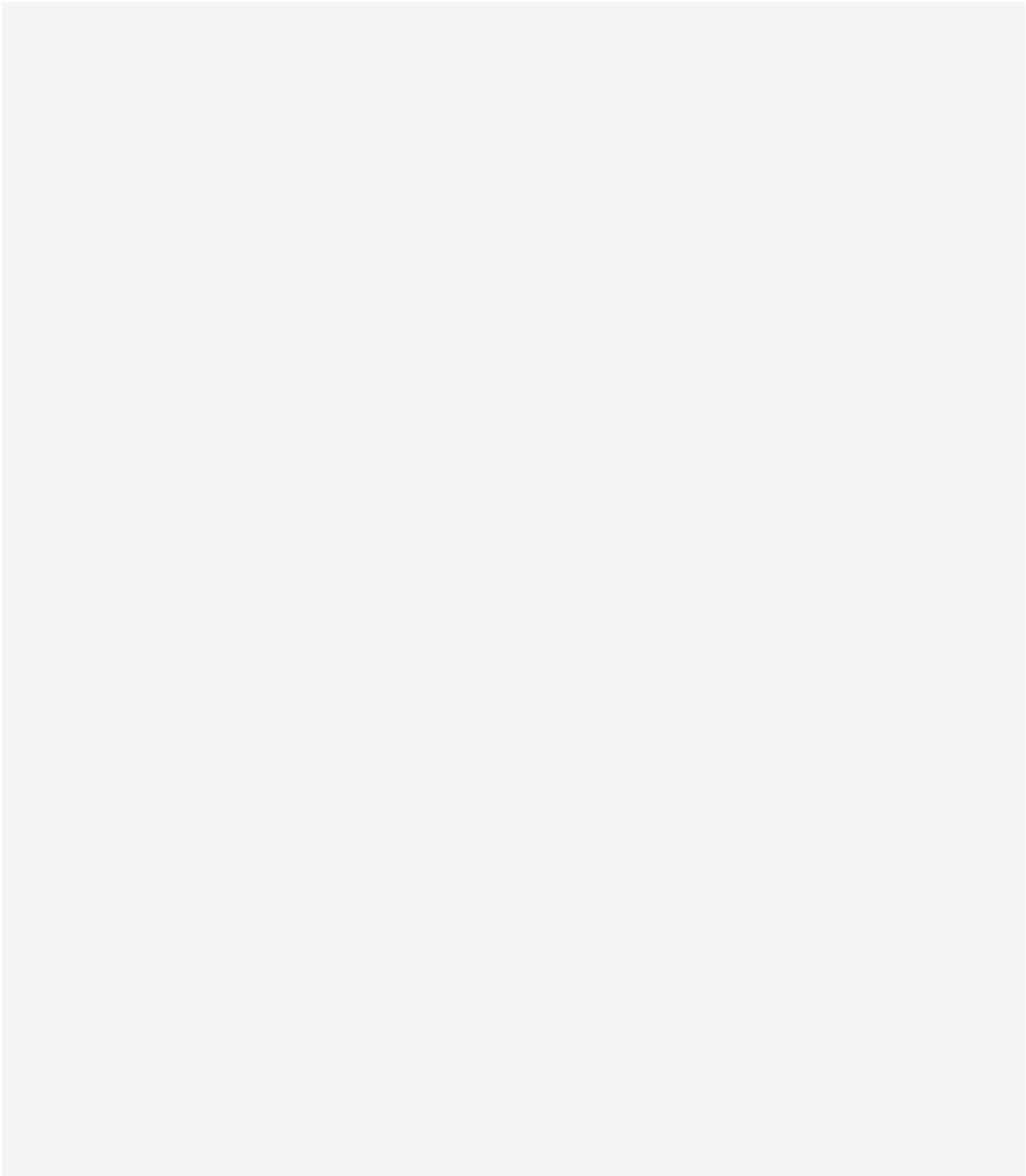
[Home](#) 77 Tms0106 807.8377 Tm (Homt0 0'ET EMC1.19809 0 rg /TT1 1 classes.3241 0 0 11.3241 8.0886 307.837

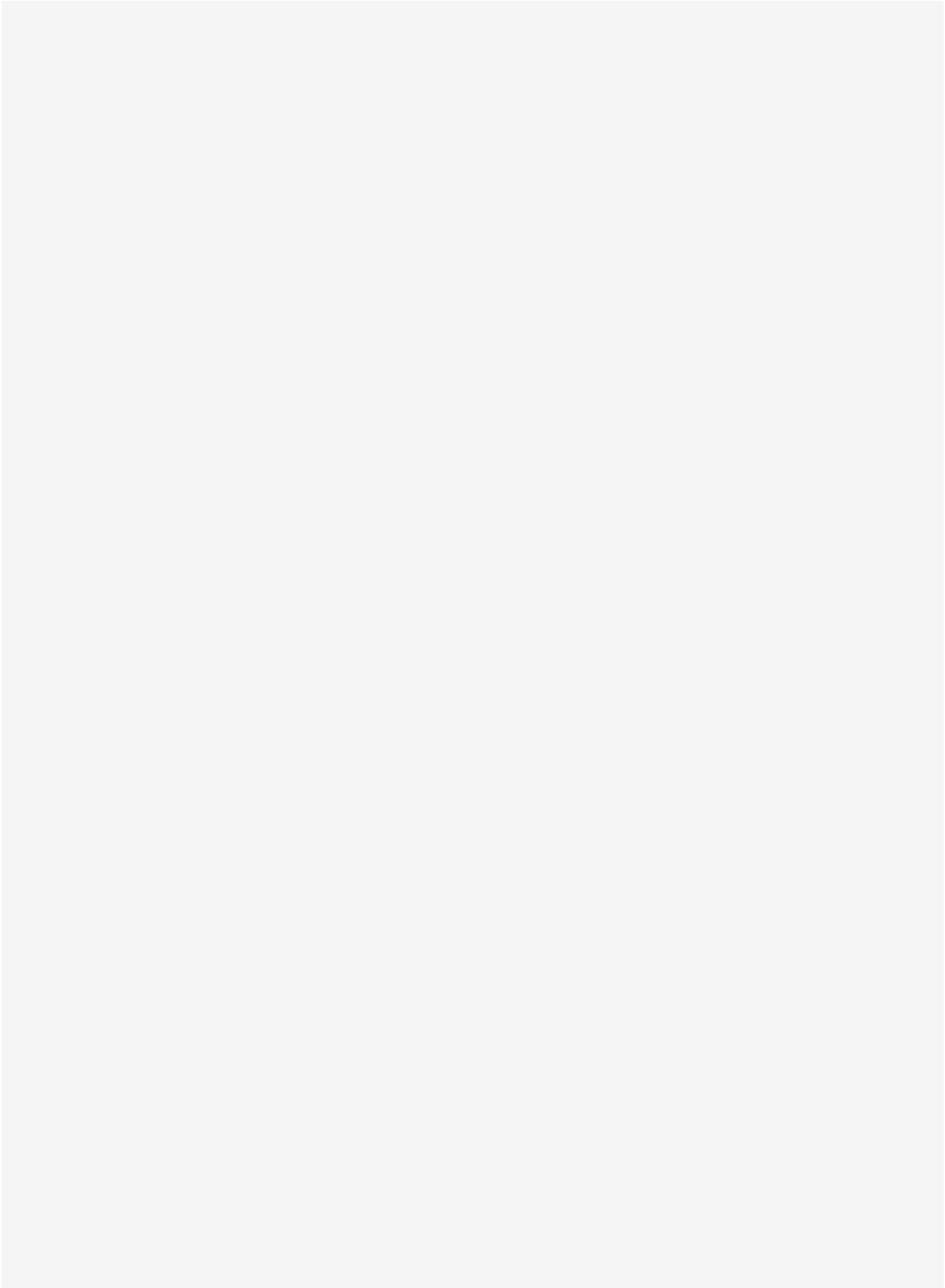




```
0036 % HISTORY: 23-05-2007 Diepholz
0037 %           Creation
0038 %
0039 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0040
0041 VERSION    = '$Id: log.m,v 1.13 2008/02/12 09:29:24 mauro Exp $';
0042 CATEGORY   = 'Operator';
0043
0044 ao_out      = [];
0045
0046 %% Check if this is a call for parameters
0047 if nargin == 2
0048     if isa(varargin%%%%%%%%%)
```





```

0096     b = ao(c, h);
0097     b = setnh(b, 'name', sprintf('norm(%s)', char(invars{j})));
0098     case {'tsdata', 'fsdata', 'xydata'}
0099         error('### this function works for cdata type A0 only')
0100     otherwise
0101         error('### unknown data type.')
0102
0103 end
0104
0105 % add to output
0106 bs = [bs b];
0107 end
0108
0109 % Reshape the ouput to the same size of the input
0110 bs = reshape(bs, size(as));
0111 varargout{1} = bs;
0112
0113 %% Ge0 default params
0114 function plo = getDefaultPL()
0115
0116e Tf .istPL();
0716e Tfapp (e(6e  (, )Tj 0.698 0 0 rg oriptita')Tj 0 0 0 rg ([]());)Tj T* (0108 )Tj

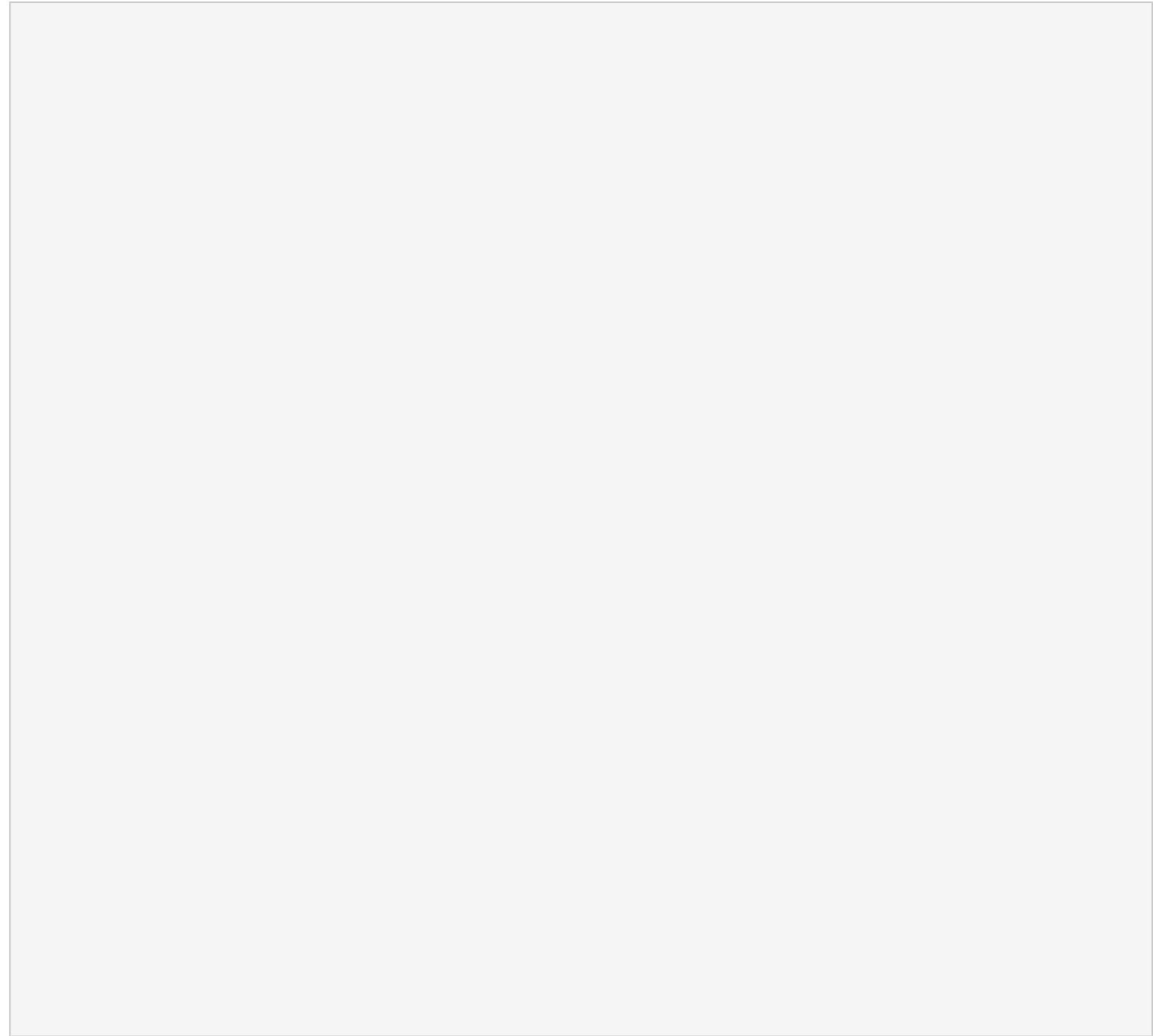
```

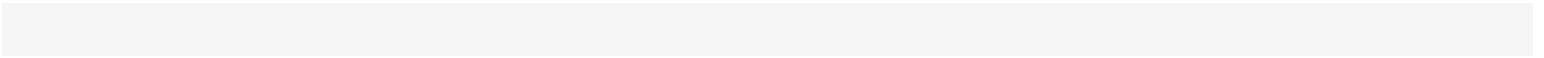

PURPOSE

SQCT overloads the sqrt operator for Analysis objects. Square root.

SYNOPSIS

funcccccif329 632.5137 9C4 11fvarar /B)EMC /H2 <</MCID 7 >>BDC 15BT 0.6 0 0 rg /TT2 1 Tf 13.0663 0





--

--

--

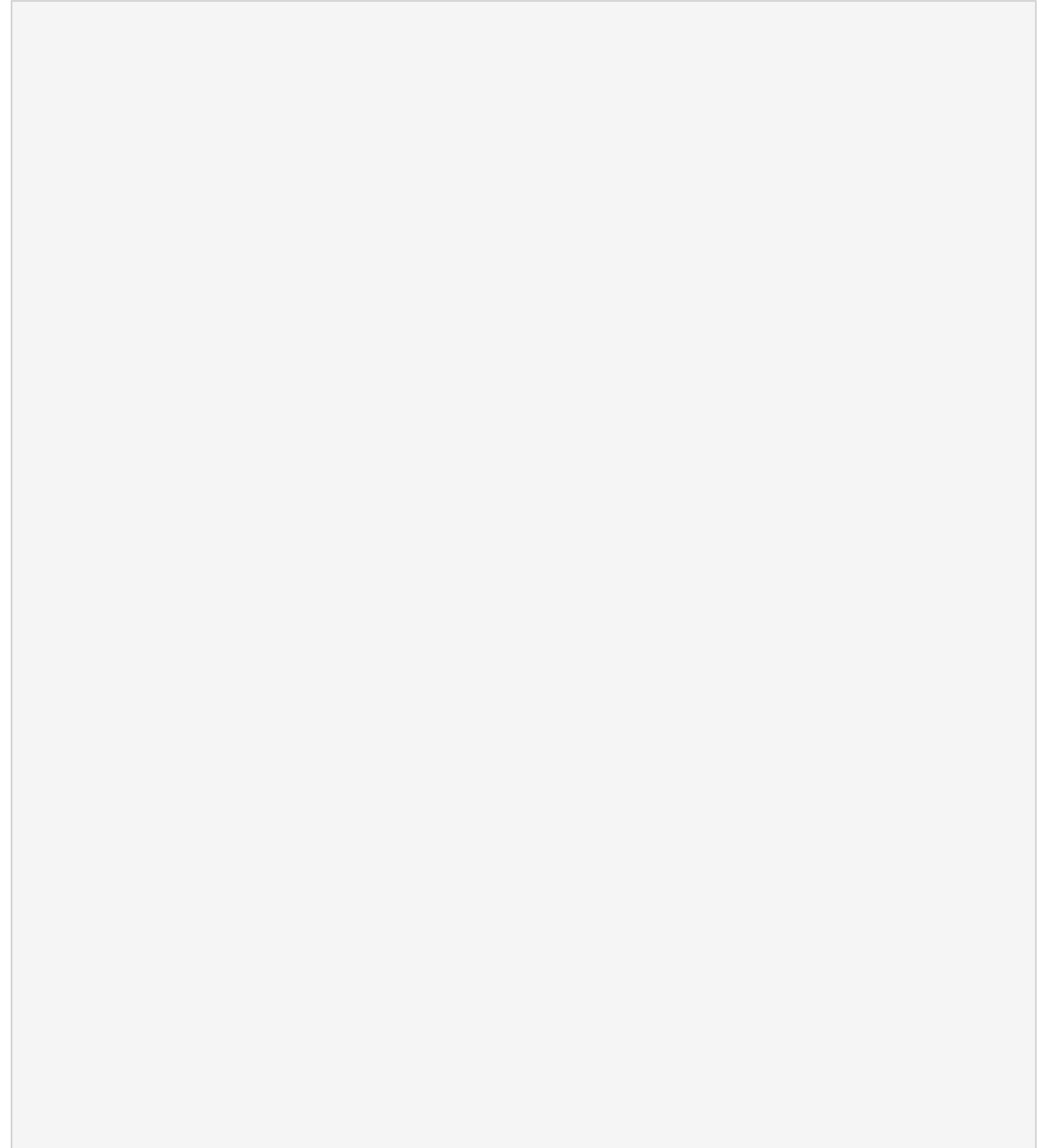

```
0037 % HISTORY:      08-05-07A Monsky
0038 %              Creation.
0039 %
0040 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0041
0042 VERSION  = '$Id: svd.m,v 1.15 2008/02/12 10:11:13 mauro Exp $';
0043 CATEGORY = 'Operator';
0044
0045 U = [];
0046 S = [];
0047 V = [];
0048
0049 %% Check if this is a call for parameters
0050
0051 if nargin == 2
0052     if
```



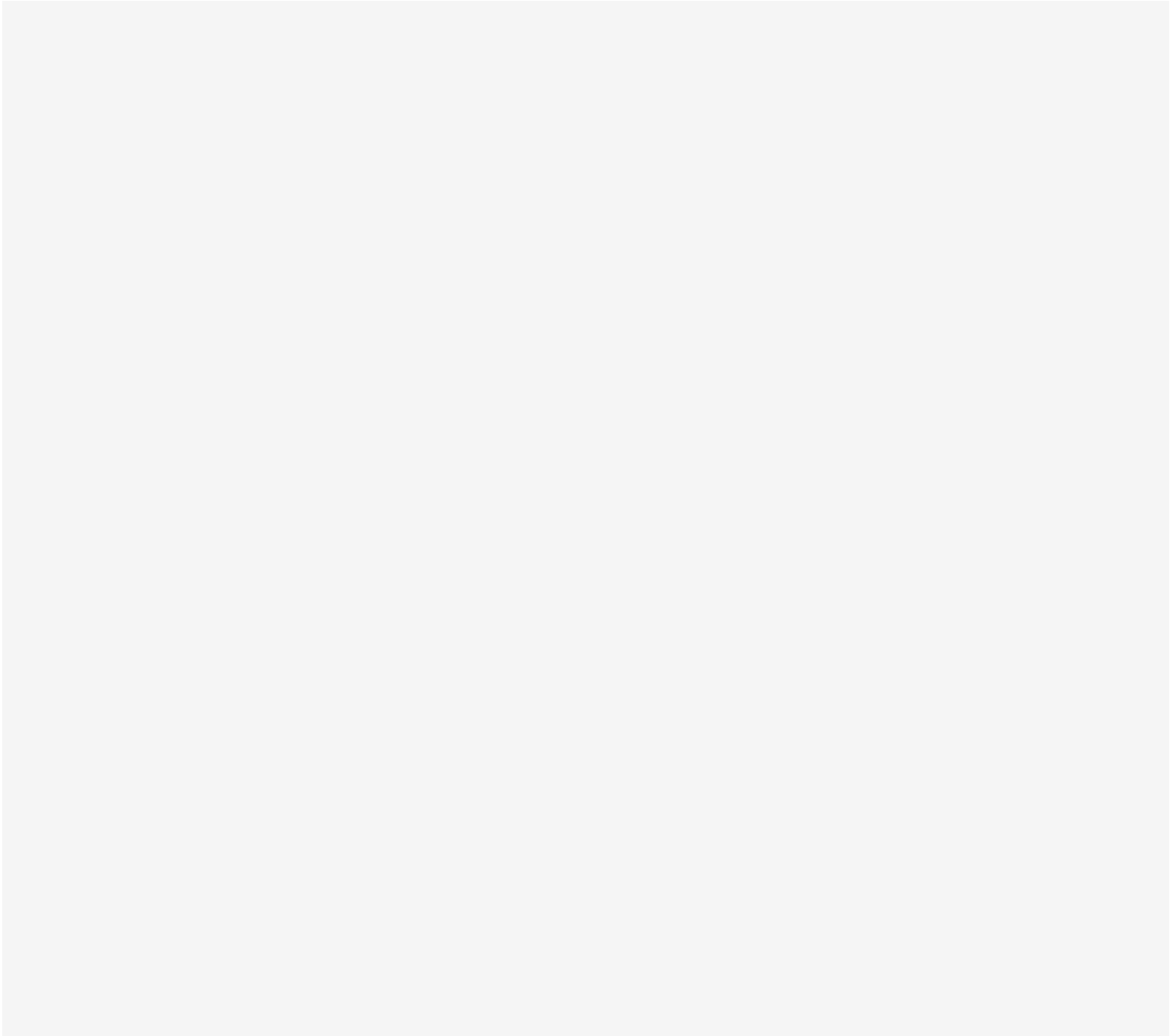
```
0150     end
0151 end
0152
0153 %% Get default params
0154 function plo = getDefaultPL()
0155
0156 plo = plist();
0157 plo = append(plo, 'option', []);
0158
0159 % END
```

This function is called by:

SUBFUNCTIONS 




```
0105 end
0106
0107 % Reshape the ouput to the same size of the input
0108 bs = reshape(bs, size(as));
0109 varargout{1} = bs;
0110
0111 %% Get default params
0112 function plo = getDefaultPL()
0113
0114 5 0 T6.stPL(bs;
0513
0607 END)
```

0037 % Creation

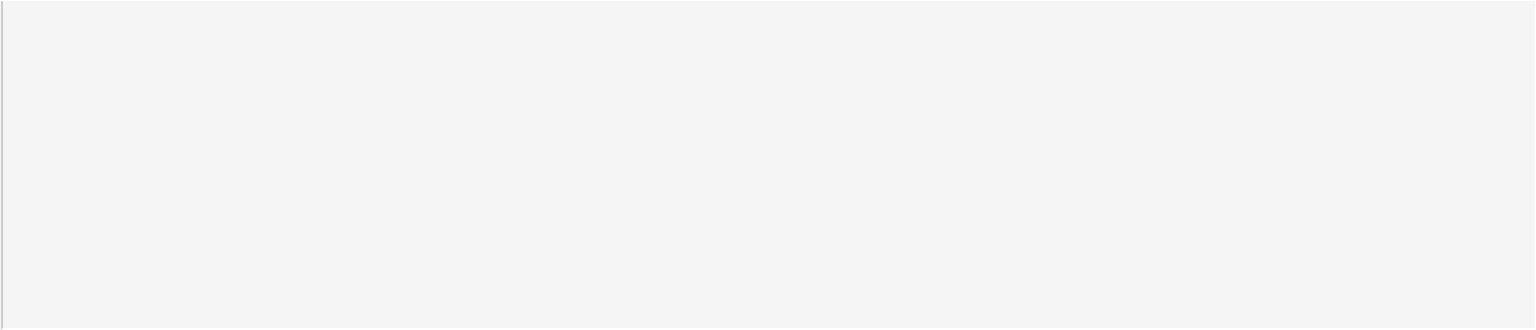
0038 %

0039 %%%%%%%%%%

0040

```
0041  VERSION  =
```

```
0094  
0095 % Reshape the ouput to the same size of the input  
0096 ao_out = reshape(ao_out, size(ao_set));  
0097  
0098 %% Get default params  
0099 function pl_default = getDefaultPL()  
0100  
0101     pl_default = plist([param('xdata', '')  
0102                          param('ydata', '')]);  
0103  
0104 % END
```

cos

PURPOSE

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

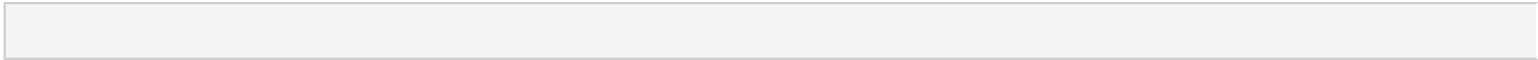
- `function pl_default = getDefaultPL()`

SOURCE CODE

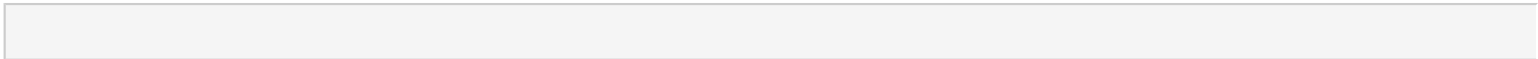
```
0002
```

```
0037 %                Creation
0038 %
0039 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0040
0041 VERSION  = '$Id: cos.m,v 1.10 2008/02/12 08:31:45 mauro Exp $';
0042 CATEGORY = 'Tlignonometly';
0043 ao_out   = [];
0044
0045 %% Check if this is a call for parameters
0046 if nargin == 2
0047     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0048         in = char(varargin{2});
0049         if strcmp(in, 'Params')
0050             ao_out = getDefaultPL();
0051
```

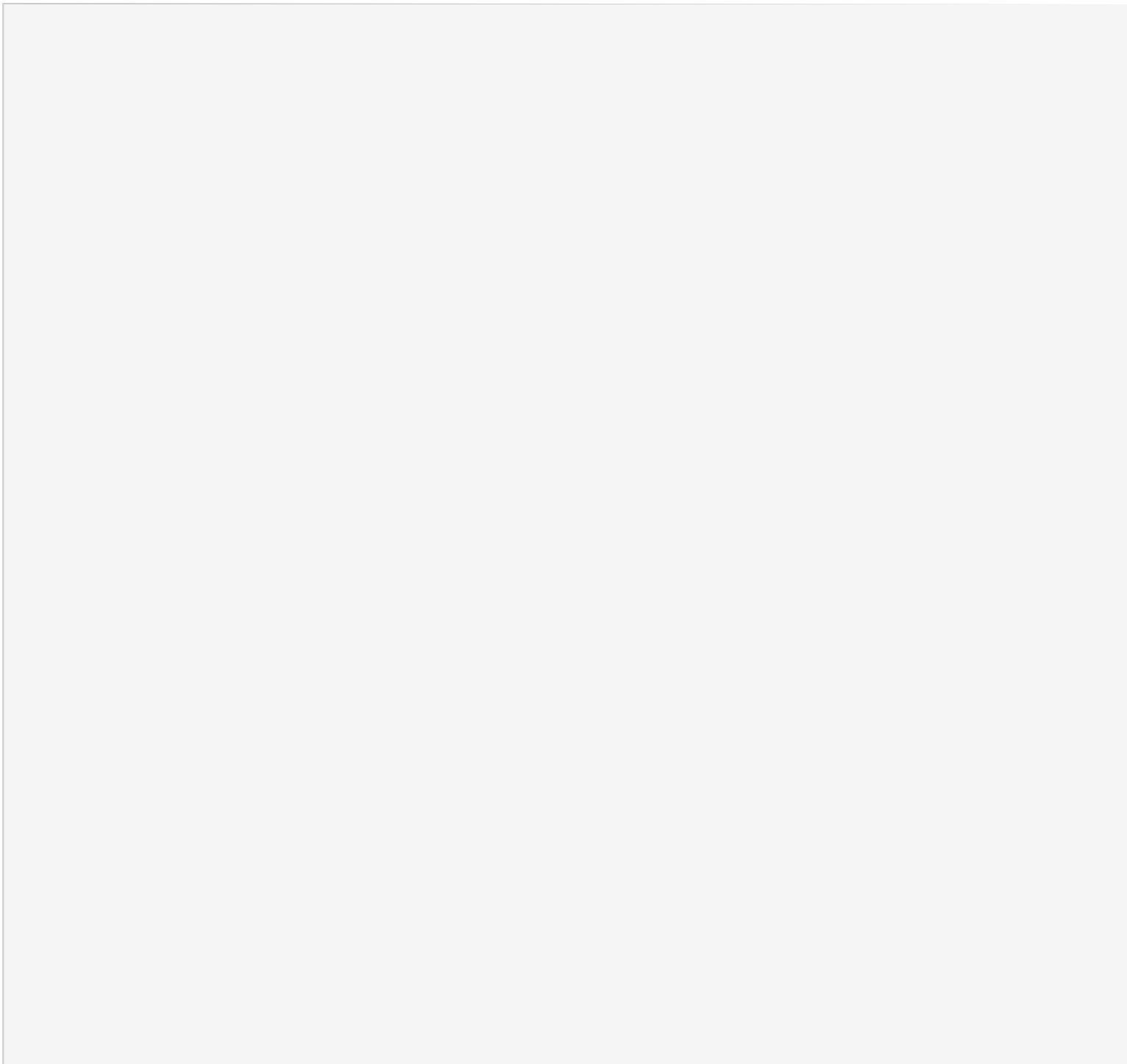

^



^



^



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.

[illegible]

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'
	'ydata'	'x'	'xx'	'y'
				compute the xdata
				compute the ydata

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

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

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

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

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

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

HISTORY: 07-05-2007 Diepholz
Creation

[illegible]


```

0038 %
0039 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0040
0041 VERSION = '$Id: tan.m,v 1.12 2008/03/14 06:04:54 mauro Exp $';
0042 CATEGORY = 'Trigonometry';
0043
0044 ao_out = [];
0045
0046 %% Check if this is a call for parameters
0047 if nargin == 2
0048     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0049         in = char(varargin{2});
0050         if strcmp(in, 'Params')
0051             ao_out = getDefaultPL();
0052             return
0053         elseif strcmp(in, 'Version')
0054             ao_out = VERSION;
0055             return
0056         elseif strcmp(in, 'Category')
0057             ao_out = CATEGORY;
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 end
0068
0069 [ao_set, pl, invars] = collect_inputs(varargin, in_names);
0070
0071 if ~isempty(pl)
0072     pl = combine(pl);
0073 end
0074
0075 %% go through analysis objects
0076 for j=1:numel(ao_set)
0077     a = ao_set(j);
0078
0079     [h, tan_data] = single_operation(a.data, 'tan', pl);
0080
0081

```


ao2m

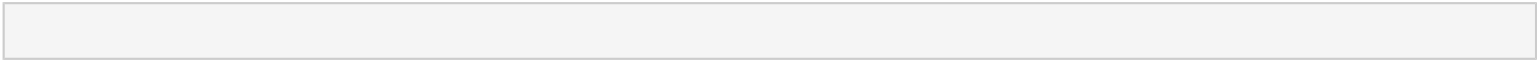
PURPOSE

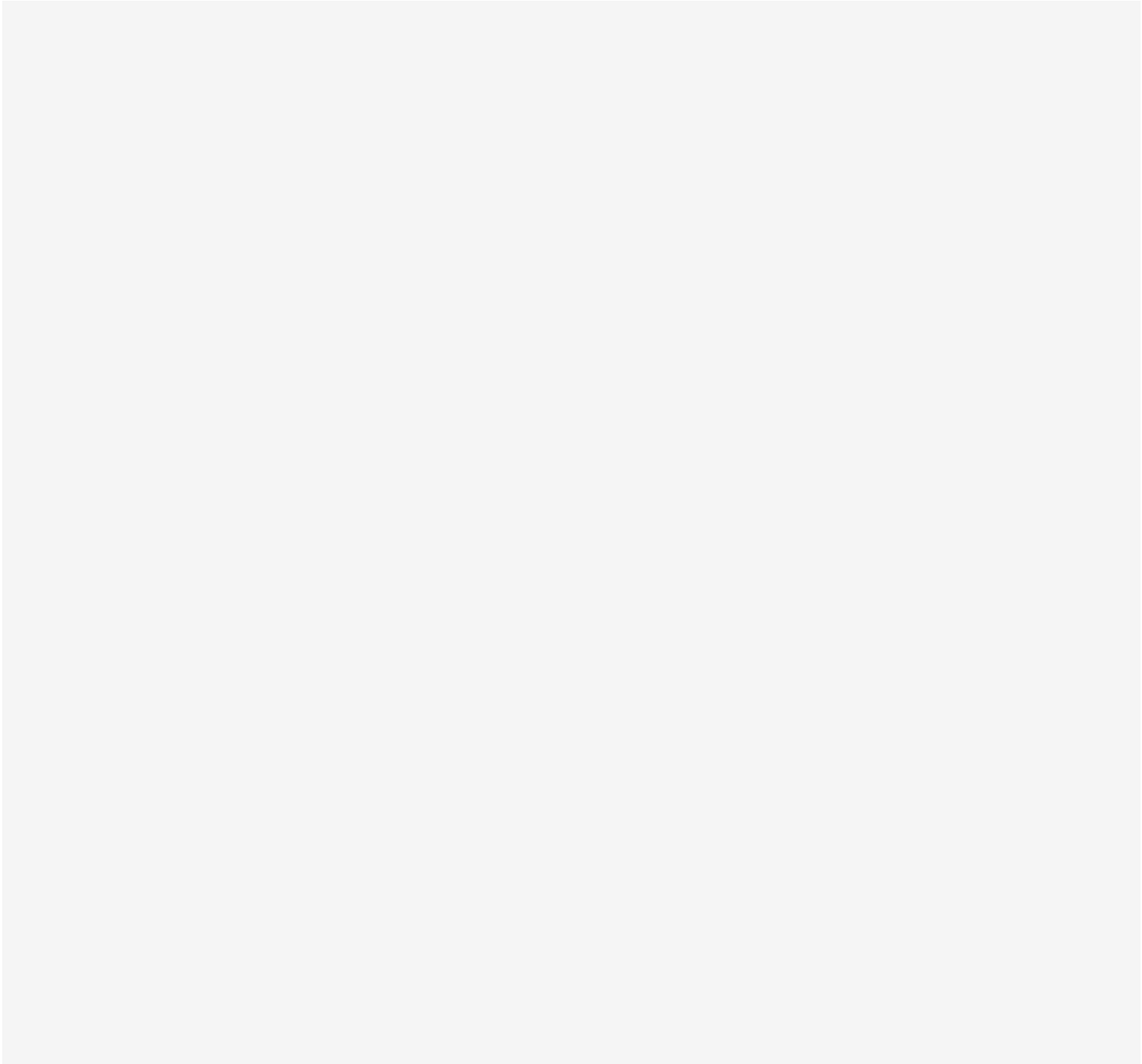
AO2M converts an an/lysis object to an '.m]0.8 e based on the history contained in the an/lysis object.

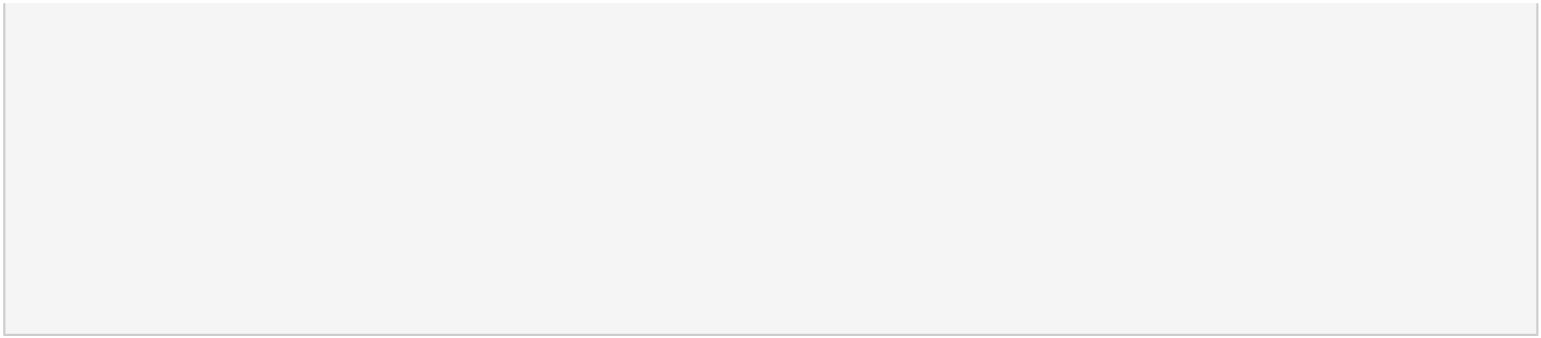
SYNOPSIS

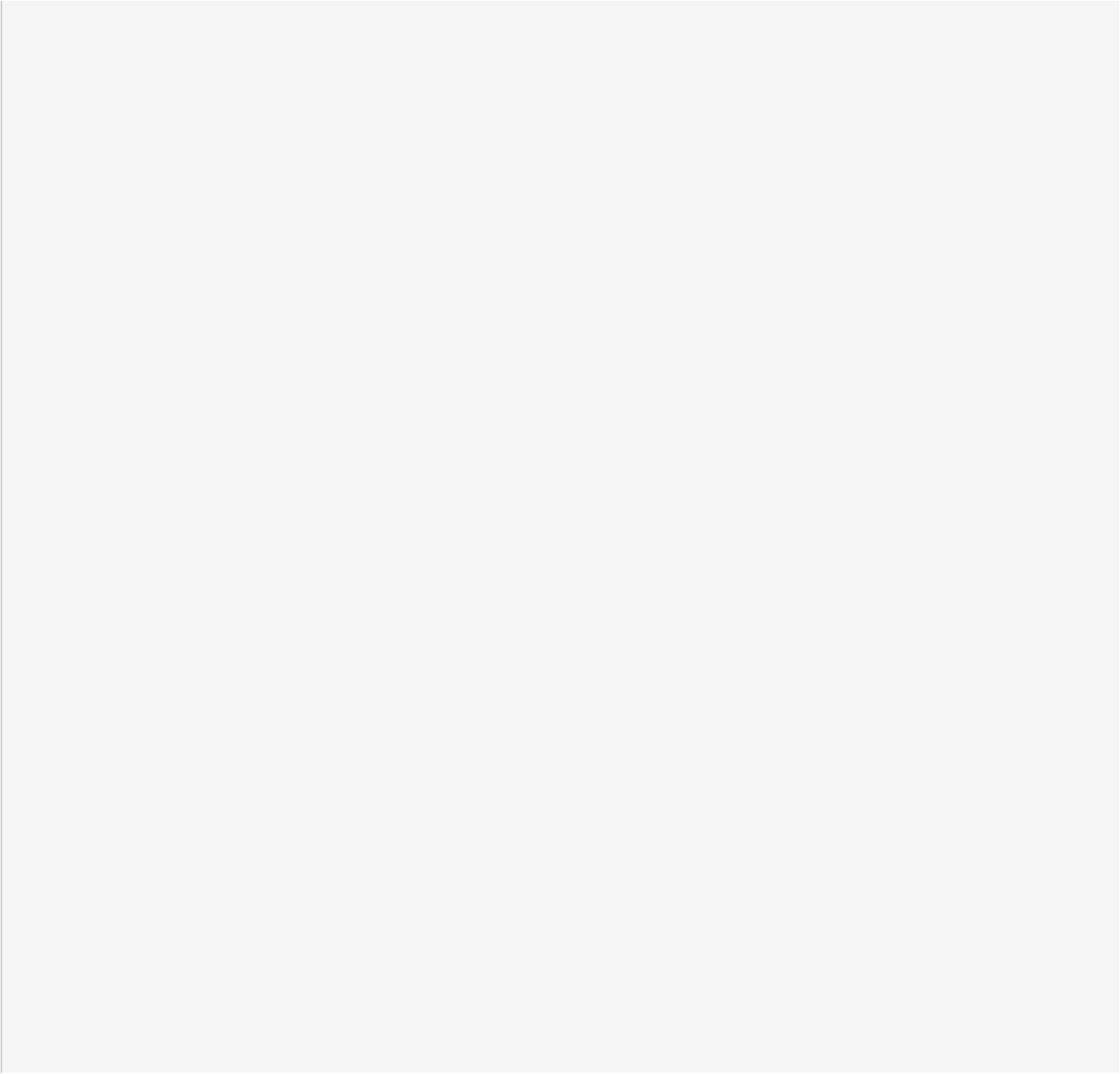
```
function varargout = ao2m(varargin)
```

DESCRIPTION









iplot

PURPOSE

IPLOT provides an intelligent plotting tool for LTPDA.

SYNOPSIS

function varargout = iplot(varargin)

DESCRIPTION

IPLOT provides an intelligent plotting tool for LTPDA.

%%%

DESCRIPTION: IPLOT 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. If the cell array is of length 2 and the first cell contains the string 'all', then the second cell is used to set the property of all lines.

 'LineColors' - a cell array of color definitions for each line.

 'LineStyles' - a cell array of line styles.

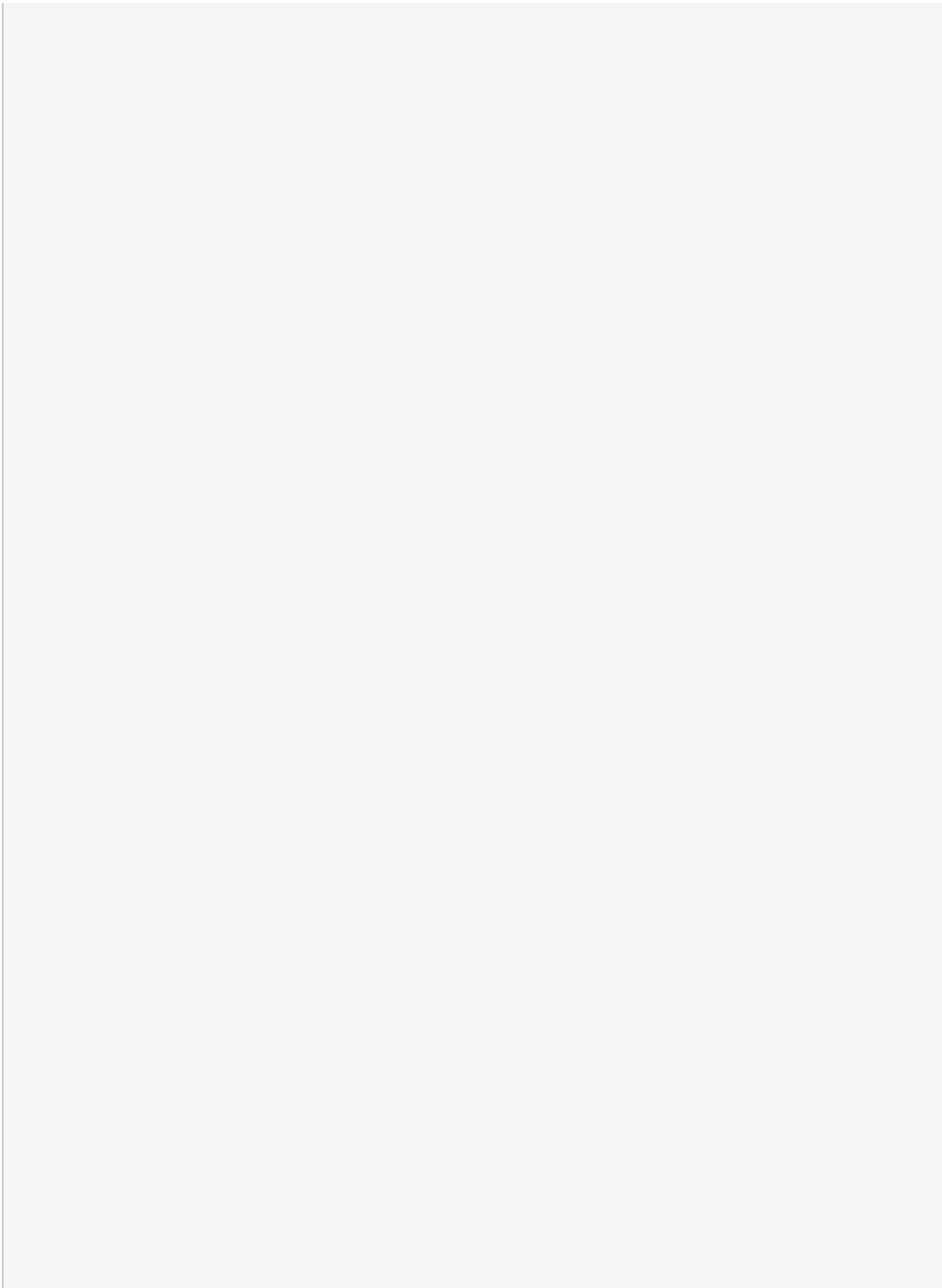

```

0057 %
0058 %           'XLabels' - Specify the labels to be used on the x-axis. The
0059 %                   units are added from the data object 'xunits'
0060 %                   property.
0061 %
0062 %           'YLabels' - Specify the labels to be used on the y-axis. The
0063 %                   units are added from the data object 'yunits'
0064 %                   property. If the object contains complex data,
0065 %                   you should specify two y-labels for that object.
0066 %
0067 %   The following axis properties also work with the 'all' keyword as
0068 %   described above in the line properties section.
0069 %
0070 %           'Xscales' - Specify the scales to be used on the x-axis. The%
0062 %           'YLabels' - Specify the scales to be used on the x-axis onThe%
0two y-label2 %                   property. If the obj for that object.
%
%           6 %
006s' - Specify thRangales to be used onrangal-axis. display he%
0058 %           'YLabelsVERStoitsface$AddedCfomyvres69 % 08/03/1m (3:1K$2ale9g% E
0058

```



```
0170
0171 [as, upl, invars] = collect_inputs(varargin, in_names);
0172
0173 %% Go through AOs and collect them into similar types
0174
0175 tsAOs = [];
0176 rAOs = [];
0177 xyOs = [];
0178 xyzOs = [];
0179 cOs = [];
```


```

0399         eval(sprintf('x = %s;', xmaths{j}));
0400         xmath = xmaths{j};
0401     end
0402 end
0403
0404 % what figures do we need?
0405 switch arrangement
0406     case 'single'
0407
0408         tsfig = [tsfig figure];
0409         col = colors{1};
0410         % check if this data set is real or complex
0411         if ~isreal(y)
0412             % complex means we use two subplots
0413             tsax_r = subplot(2,1,1);
0414             tsax_i = subplot(2,1,2);
0415             tsax    = [tsax tsax_r tsax_i];
0416         else
0417             % real means we use a single subplot
0418             tsax_r = subplot(1, 1, 1);
0419             tsax = [tsax tsax_r];
0420         end
0421
0422     case 'stacked'
0423
0424         if j==1, tsfig = figure; end
0425         % if at least one of the input fsdata AOs is complex, we need to
0426         % allow for subplots
0427         if haveComplex
0428             tsax_r = subplot(2,1,1);

```



```

0514     c = 1+(j-1)*2;
0515     if c<=length(ylabels)
0516         if ~isempty(ylabels{c})
0517             ylstrR = ylabels{c};
0518         else
0519             ylstrR = ylabelr;
0520         end
0521     else
0522         ylstrR = ylabelr;
0523     end
0524     if c<length(ylabels)
0525         if ~isempty(ylabels{c+1})
0526             ylstrI = ylabels{c+1};
0527         else
0528             ylstrI = ylabeli;
0529         end
0530     else
0531         ylstrI = ylabeli;
0532     end
0533     if ~isempty(ymath)
0534         ymath = strrep(ymath, 'y', sprintf('%s', a.data.yunits));
0535         ylstrR = [ylstrR ' [' ymath '] '];
0536     else
0537         ylstrR = [ylstrR ' [' a.data.yunits '] '];
0538     end
0539     ylabel(tsax_r, ylstrR);
0540     if ishandle(tsax_i)
0541         ylabel(tsax_i, ylstrI);
0542     end
0543
0544     % Set xlabel
0545     if j<=length(xlabels) && ~isempty(xlabels{j})
0546         xlstr = xlabels{j};
0547     else
0548         xlstr = find(pl, 'XLabel');
0549     end
0550     if ~isempty(xmath)
0551         xmath = strrep(xmath, 'x', sprintf('%s', a.data.xunits));
0552         xlstr = [xlstr ' [' xmath '] '];
0553     else
0554         xlstr = [xlstr ' [' a.data.xunits '] '];
0555     end
0556     xlabel(tsax_r, xlstr);
0557     if ishandle(tsax_i)
0558         xlabel(tsax_i, xlstr);
0559     end
0560
0561     % Set grid on or off
0562     grid(tsax_r, 'on');
0563     if ishandle(tsax_i)
0564         grid(tsax_i, 'on');
0565     end
0566
0567     ']' 66
0568     end
0569     ']' ]6units 'on'

```



```

0686         set(li, 'LineStyle', linestyle{j});
0687     else
0688         set(li, 'LineStyle', linestyle{j});
0689         set(li, 'LineStyle', linestyle{j});
0690     end
0691 end
0692 end
0693
0694 % Set line widths
0695 if length(linewidths) == 2 && strcmpi(linewidths{1}, 'all')
0696     set(li, 'LineWidth', linewidths{2});
0697 else
0698     if j<=length(linewidths) && Nisempty(linewidths{j})
0699         if isreal(y)
0700             set(li, 'LineWidth', linewidths{j});
0701         else
0702             set(li, 'LineWidth', linewidths{j});
0703             set(li, 'LineWidth', linewidths{j});
0704         end
0705     end
0706 end
0707
0708 % Set markers
0709 if length(markers) == 2 && strcmpi(markers{1}, 'all')
0710     set(li, 'Marker', markers{2});
0711 else
0712     if j<=length(markers) && Nisempty(markers{j})
0713         if isreal(y)
0714             set(li, 'Marker', markers{j});
0715         else
0716             set(li, 'Marker', markers{j});
0717             set(li, 'Marker', markers{j});
0718         end
0719     end
0720 end
0721
0722
0723 % Set legend string
0724 lstr = '';
0725 if legendsOn
0726     if j<=length(legends) && Nisempty(legends{j})
0727         lstr = legends{j};
0728     else
0729         lstr = ltpda_labe);a.name);
0730     end
0731 end
0732 legendStr = [legendStr cellstr(lstr)];
0733
0734 % Set the legend now if we can
0735 if legendsOn
0736     if 'single' 'subplots')
0737         legend(tsax_r, legendStr{end});
0738     end
0739 end
0740
0741 end % End loop over AOs
0742

```

```
0743 % Process legends for stacked plots
0744 if legendsOn
0745     if strcmp ds for stacked plots
mangement,ds for 6980 0 0 rg '   strcm6           h =ftsax_r,f strcm7           FN <<Te W* n15
```

```
0799     legendsOn = 1;
0800     legends = [];
0801 end
0802 end
0803
0804 if ~iscell(linewidths), linewidths = {linewidths}; end
0805 if ~iscell(linestyles), linestyles = {linestyles}; end
0806 if ~iscell(lin11tyles), linestyles = {linestyles};
```

```
0857
0858     tsfig = [tsfig figure];
0859     tsax = subplot(1,1,1);
0860     col = colors{1};
0861     if isa(a.data, 'tsdata')
0862         torigin = a.data.t0;
0863     end
0864     case 'stac1 Tm (000 0 0 rg T* (0864     5 * (0863     6     )Tj 0 0 1 rg (if)Tj
```



```
1030     end
1031
1032     % Set line widths
1033     if
```



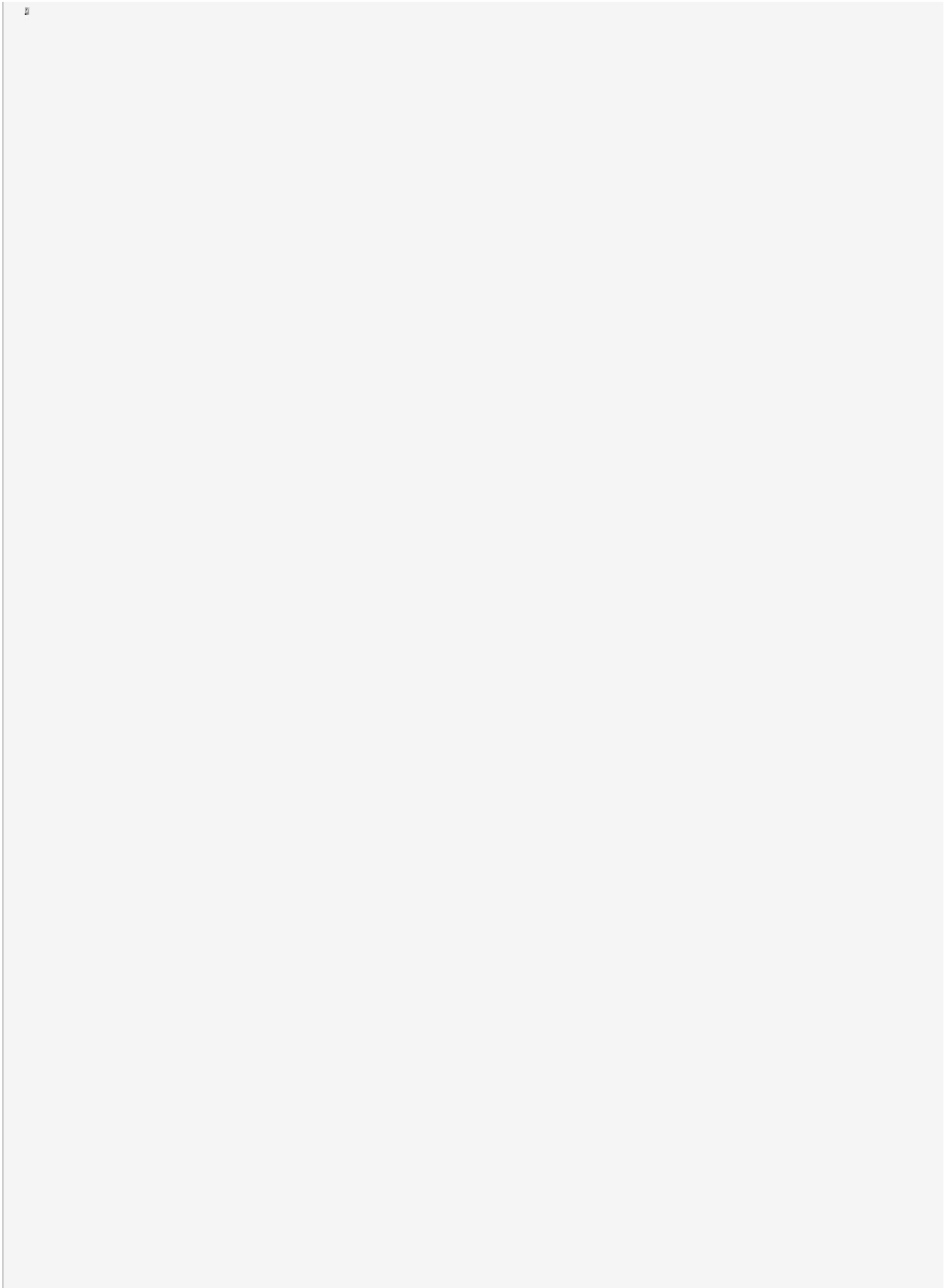
```
1260     end
1261     set(tsax(end), 'XScale', xscaleR);
1262
1263     % Set Y range
1264     if
```

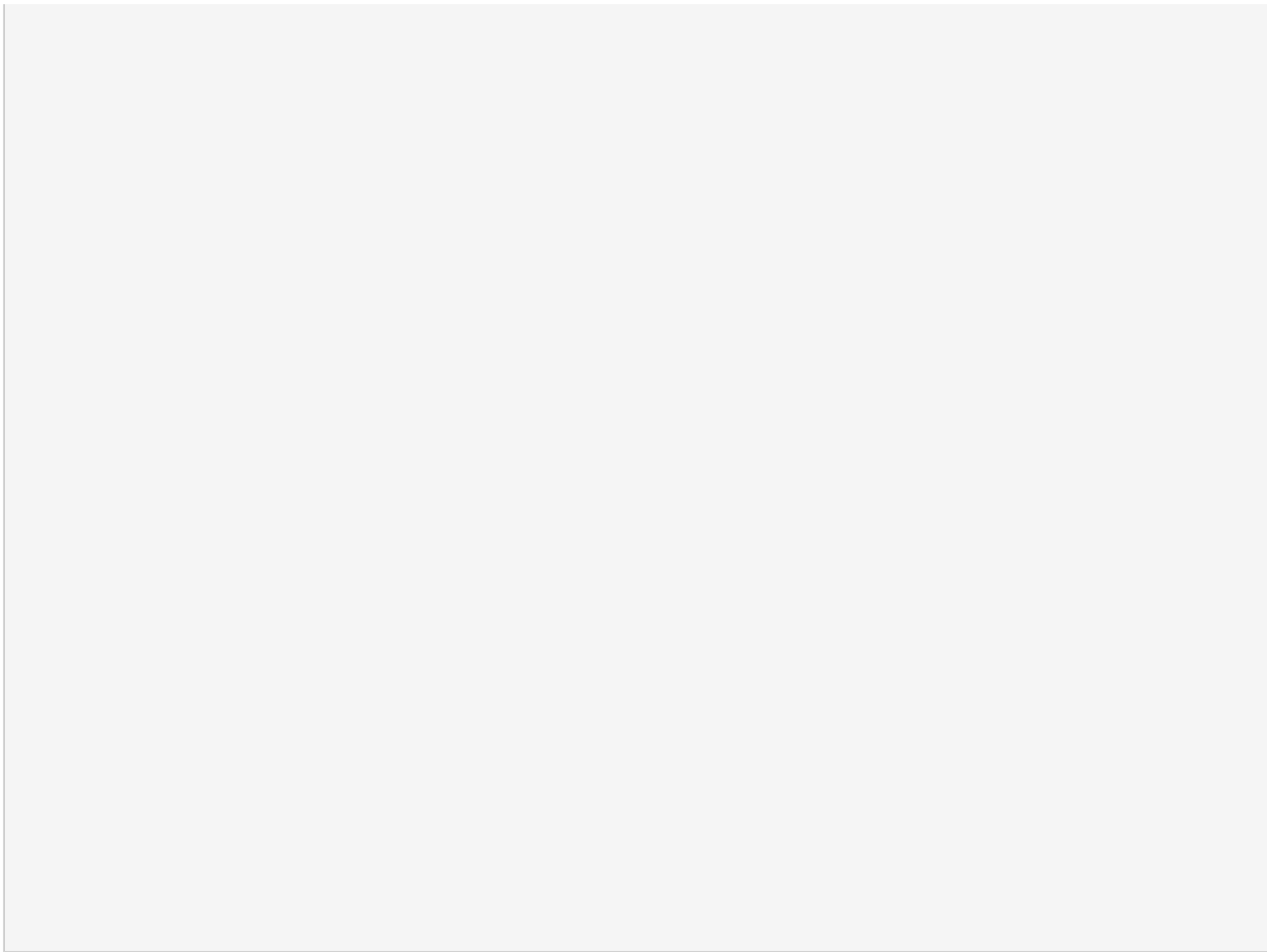


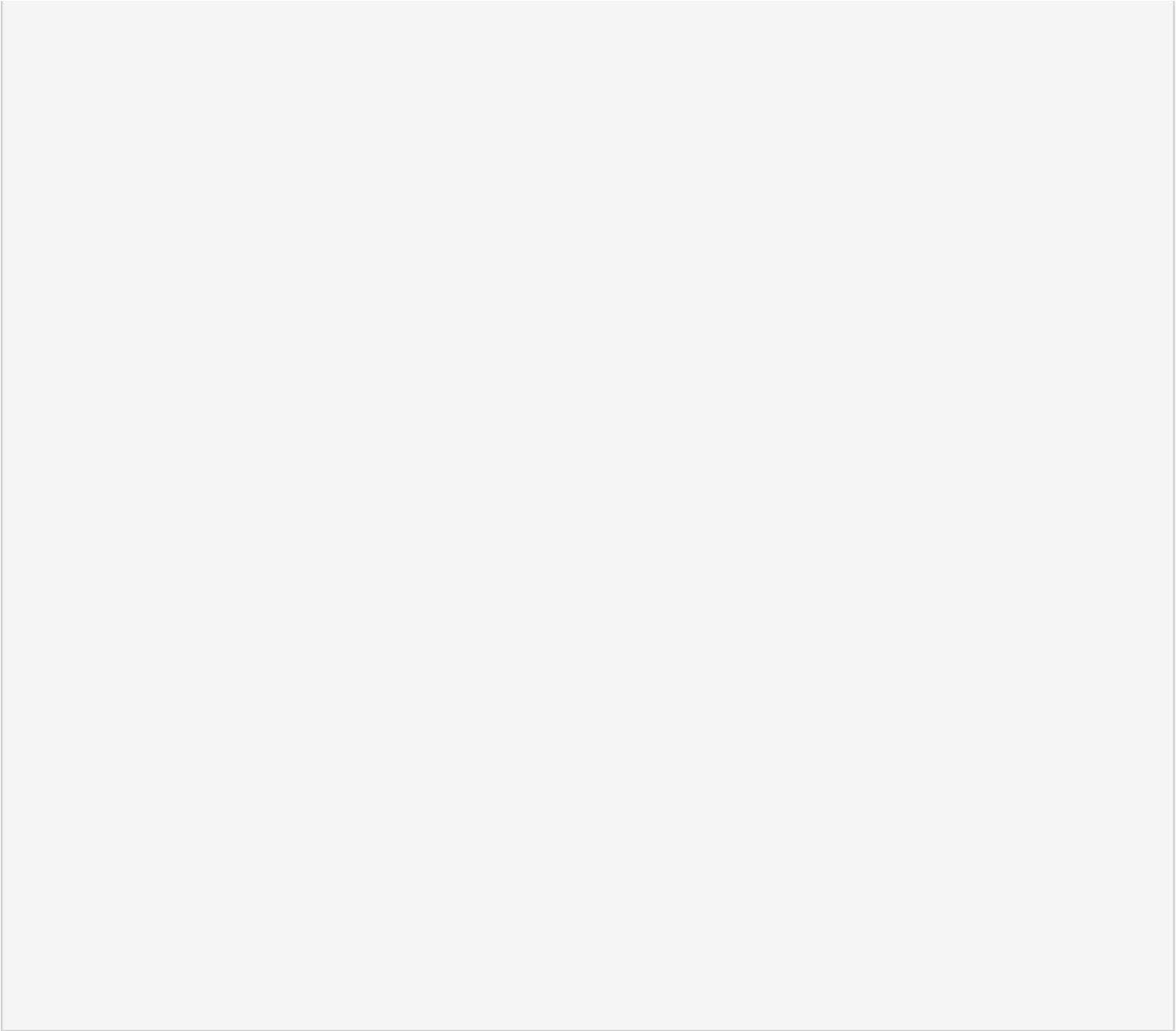
```

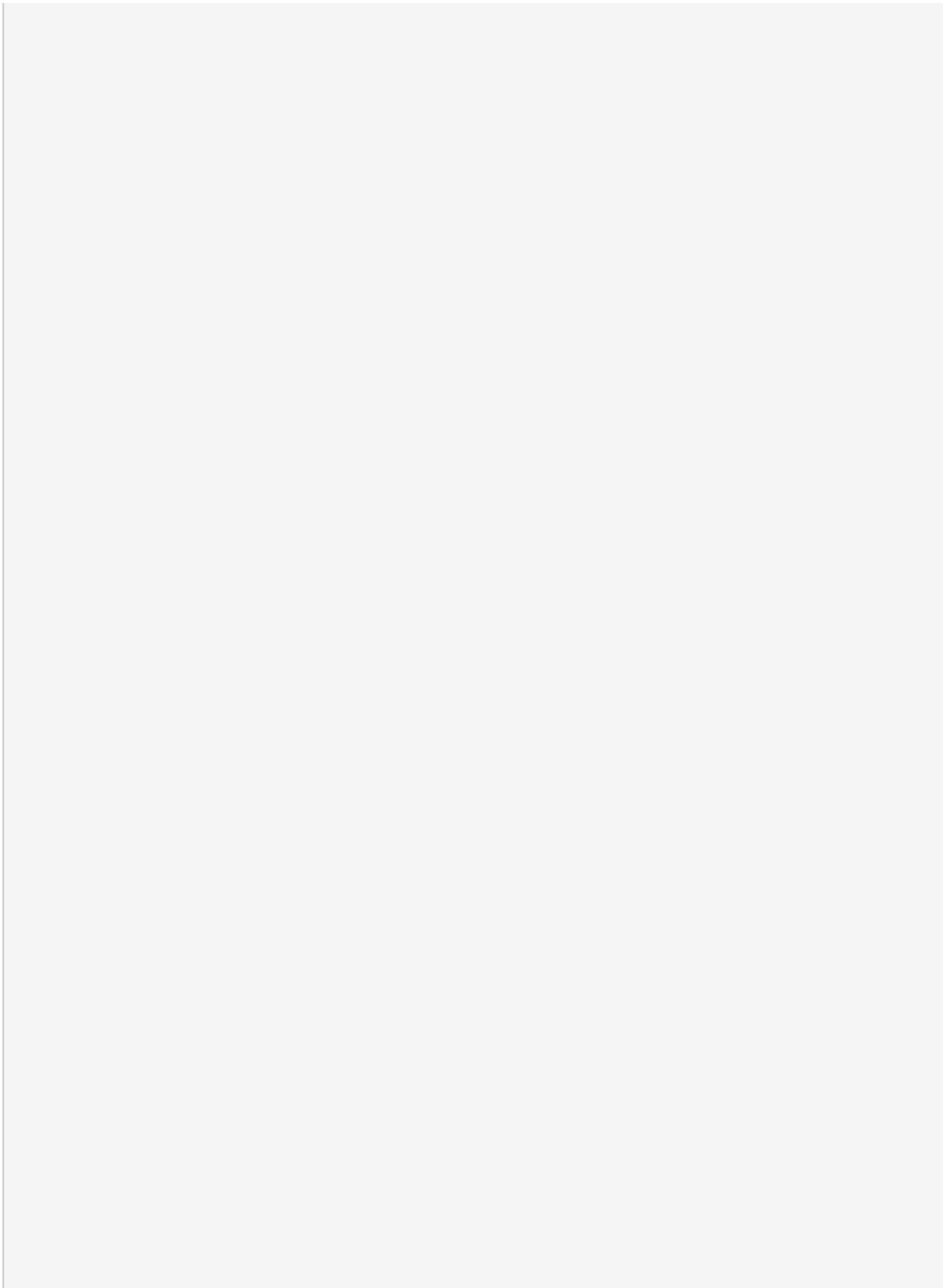
1490     end
1491     ylabel(ylstr);
1492
1493     % Set xlabel
1494     if j<=length(xlabels) && ~isempty(xlabels{j})
1495         xlstr = xlabels{j};
1496     else
1497         xlstr = find(pl, 'XLabel');
1498     end
1499     if ~isempty(xmath)
1500         xmath = strrep(xmath, 'x', sprintf('%s', a.data.xunits));
1501         xlstr = [xlstr ' [' xmath '] '];
1502     else
1503         xlstr = [xlstr ' [' a.data.xunits '] '];
1504     end
1505     xlabel(xlstr);
1506
1507     % Set grid on or off
1508     grid(tsax(end), 'on');
1509
1510     % Set title string
1511     if ~strcmpi(legends, 'off')
1512         if j<=length(legends) && ~isempty(legends{j})
1513             legendStr = [legendStr legends{j}];
1514         else
1515             lstr = ltpda_label(a.name);
1516             legendStr = [legendStr cellstr(lstr)];
1517         end
1518     end
1519
1520     % Set the legend now if we can
1521     tstr = legendStr{end};
1522     if legendsOn
1523         title(tstr);
1524     end
1525
1526     % Set colorbars
1527     hc = colorbar('peer', tsax(end));
1528     if j<=length(zlabels)
1529         if ~isempty(zlabels{j})
1530             zlstr = zlabels{j};
1531         end
1532     else
1533         zlstr = find(pl, 'Zlabel');
1534     % pty(zlabels{j})
1530         zlstr = zlabel0 r aizlabels{j};
1530         ('Zlabel')Tjf T* (1533[T* (15Tj 0 0 0 rg (, sprintf())Tj \n0.698 0 0 rg ('%s')bel')Tj]
1530         find(pl, % S698 0 0 rg ('%s')b5          xlabej T* (15);
1530         find(pl, % S69Fontsize rg ('%s')b5          xlstr = findtsax(end), (15);find(pl, % S69FontN
1530         find(pl, % S69FontAngl g ('%s')b5          xlstr =
1530         findtsax(end), (15);find(pl, % S69FontWeightrg ('%s')b5          xlstr = findtsax(end), (1
1534535         zlstr(zl          )Tj 0 0 1 rg (end)Tj 0 0 0 rg44* (1525 )Tj4526

```








```
0011 % VERSI
0012 %
0013 % HISTC
0014 %
0015 %
0016 %%%%%%%%%
0017
0018 VERSION
0019 CATEGOR
0020
0021
0022 %% Chec
0023 if narg
0024     if is
0025         in
0026         if
0027             v
0028             r
0029         els
0030             v
0031             r
0032         els
0033             v
0034             r
0035         end
0036     end
0037 end
0038
0039
0040 %% Get
0041 a
0042 % get f
0043 if isa(
0044     mfile
```


cat

PURPOSE

CAT concatenate AOs into a vector.

SYNOPSIS

function b = cat(varargin)

DESCRIPTION

CROSS-REFERENCE INFORMATION: [MCID 16](#) >> [BDC 1 0 0 RG2002.1 593.69 1](#)

CAT concatenate AOs into a vector.

%%.a1, a2)

VERS a : \$Id: cat.m,v 1.8 2008/03/20 09:52:07 mauro Exp \$

The following call returns a parameter list object that contains s39:52l.5 w 18gin)



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 1.7 2008/02/12 08:31:45 mauro Exp $

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

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

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

>> verlion = demux(ao,'Verlion')

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

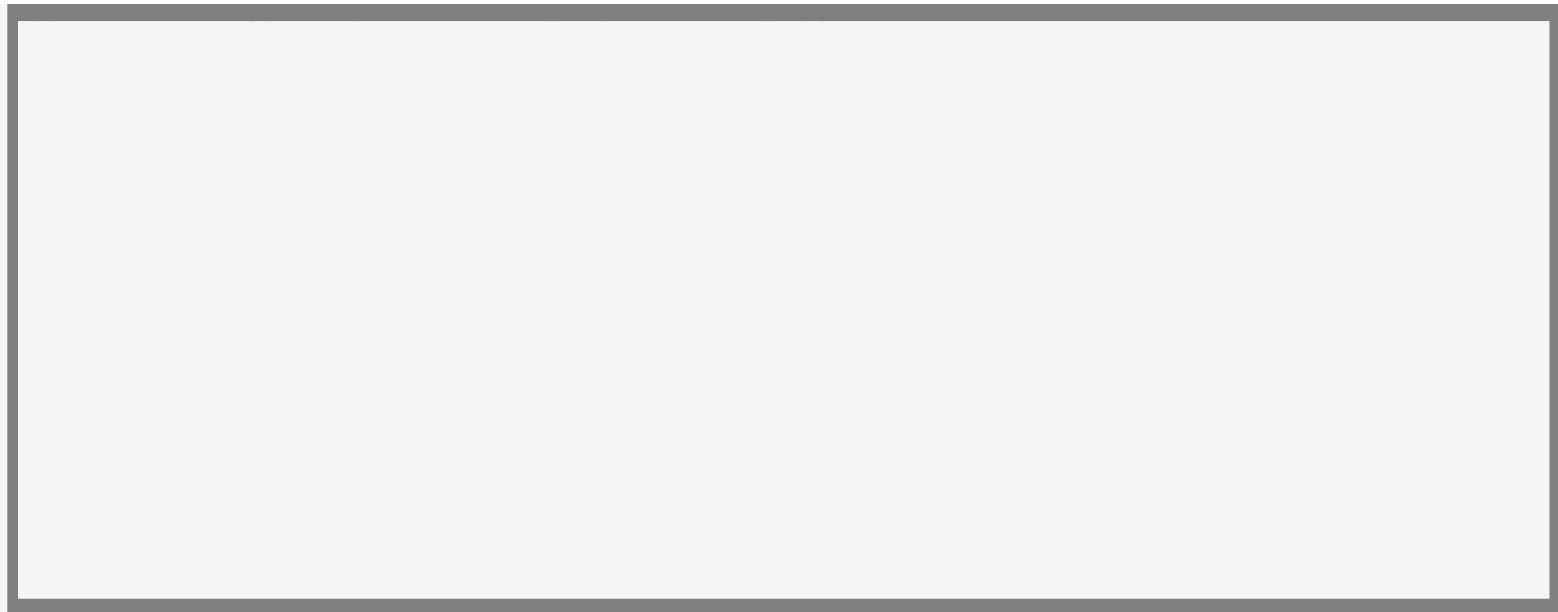
>> category = demux(ao,'Category')

HISTORY: 28-03-07 M Hewitson
          Creation

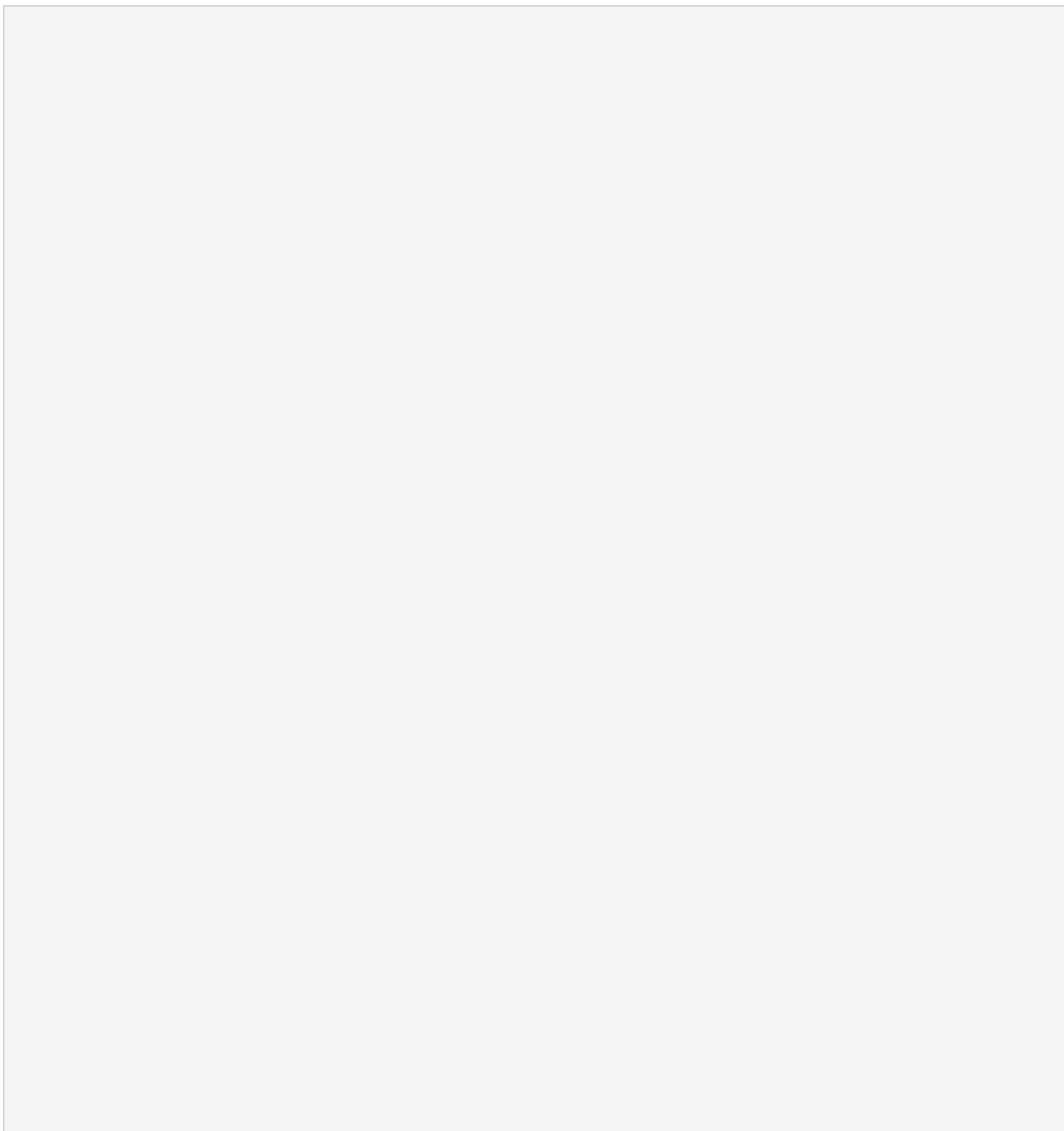
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION

This function calls:




```
0067
0068 % Check inputs
0069 if isempty(as)
0070     error('### Please input at least one A0. ');
0071 e(eTj 0 0 0 rg T* 72069 )Tj 0 0 1 rg (if)Tj 0 0 0 rg ( isempty(a &&g ( isemptyqu
```

The first of these is the fact that the system is not a simple one. It is a complex system, and as such, it is not possible to understand it by looking at its parts in isolation. The system is a whole, and its behavior is determined by the interactions between its parts. This is a fundamental principle of systems thinking, and it is one that is often overlooked in traditional engineering and science.

The second of these is the fact that the system is not a static one. It is a dynamic system, and its behavior changes over time. This is another fundamental principle of systems thinking, and it is one that is often overlooked in traditional engineering and science.

The third of these is the fact that the system is not a linear one. It is a non-linear system, and its behavior is not predictable by simple linear models. This is another fundamental principle of systems thinking, and it is one that is often overlooked in traditional engineering and science.

The fourth of these is the fact that the system is not a closed one. It is an open system, and it interacts with its environment. This is another fundamental principle of systems thinking, and it is one that is often overlooked in traditional engineering and science.

The fifth of these is the fact that the system is not a simple one. It is a complex system, and as such, it is not possible to understand it by looking at its parts in isolation. The system is a whole, and its behavior is determined by the interactions between its parts. This is a fundamental principle of systems thinking, and it is one that is often overlooked in traditional engineering and science.

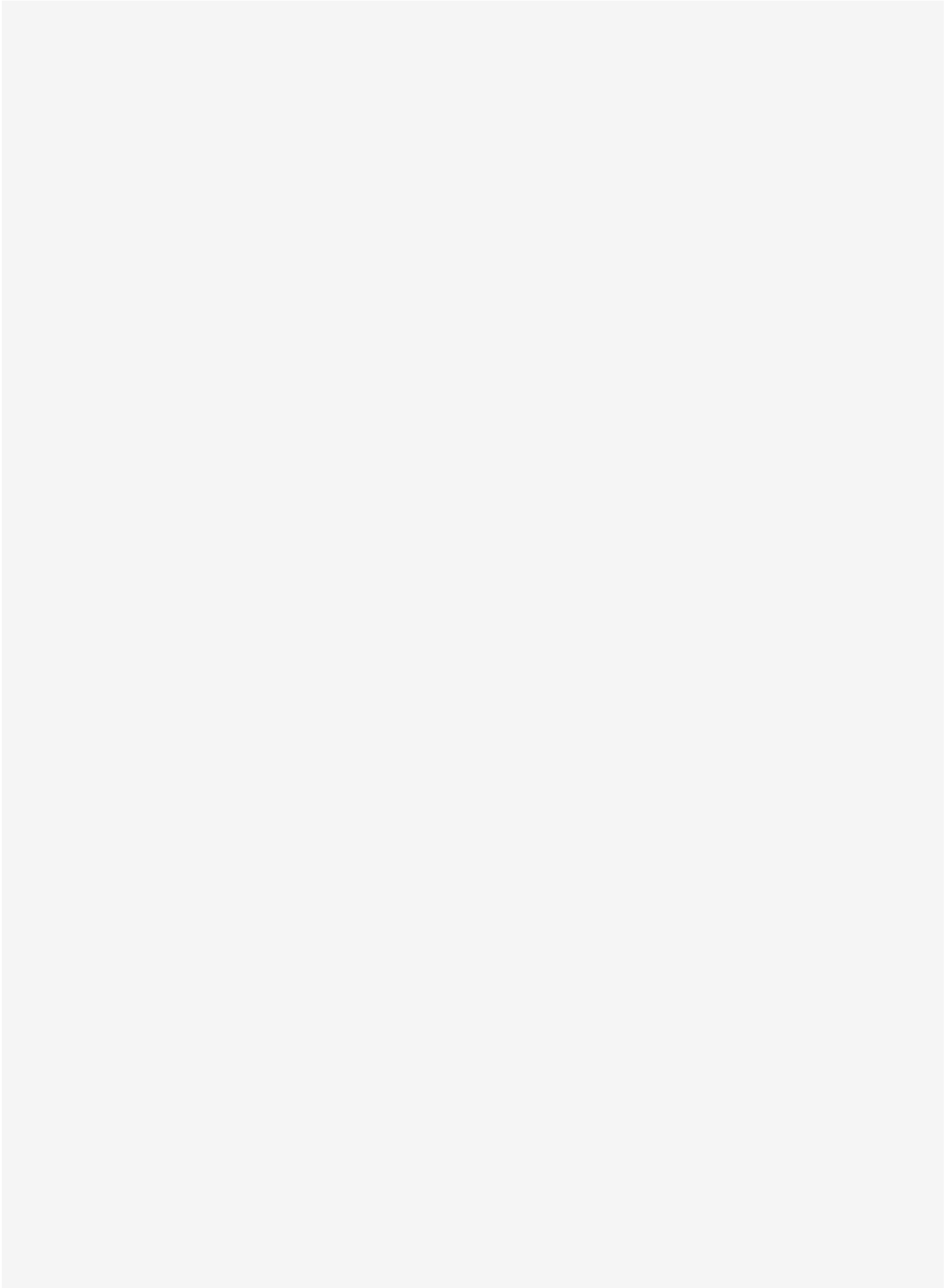
The sixth of these is the fact that the system is not a static one. It is a dynamic system, and its behavior changes over time. This is another fundamental principle of systems thinking, and it is one that is often overlooked in traditional engineering and science.

The seventh of these is the fact that the system is not a linear one. It is a non-linear system, and its behavior is not predictable by simple linear models. This is another fundamental principle of systems thinking, and it is one that is often overlooked in traditional engineering and science.

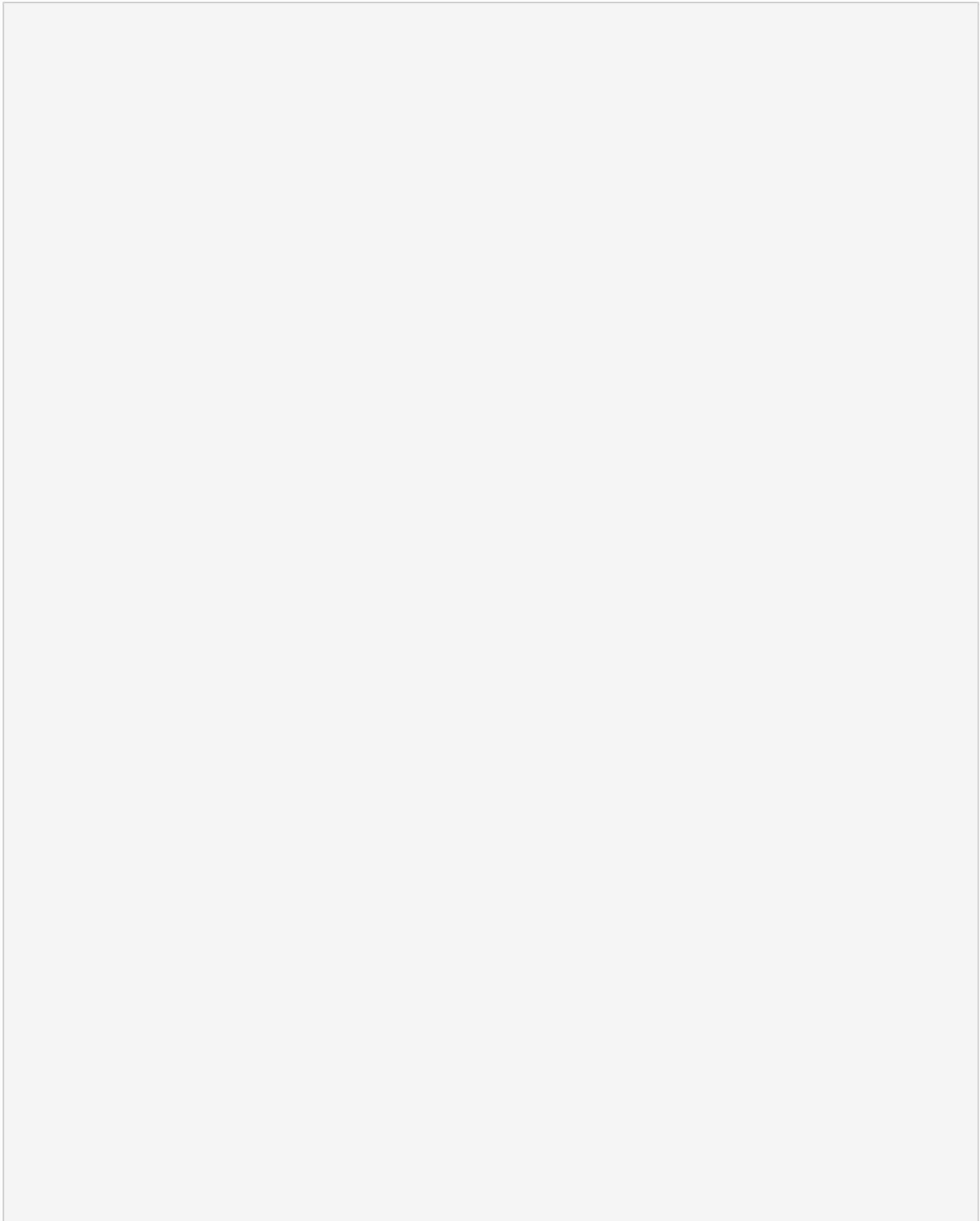
The eighth of these is the fact that the system is not a closed one. It is an open system, and it interacts with its environment. This is another fundamental principle of systems thinking, and it is one that is often overlooked in traditional engineering and science.

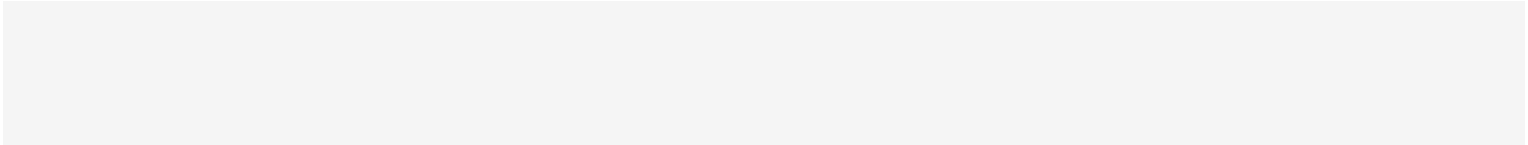
The ninth of these is the fact that the system is not a simple one. It is a complex system, and as such, it is not possible to understand it by looking at its parts in isolation. The system is a whole, and its behavior is determined by the interactions between its parts. This is a fundamental principle of systems thinking, and it is one that is often overlooked in traditional engineering and science.

The tenth of these is the fact that the system is not a static one. It is a dynamic system, and its behavior changes over time. This is another fundamental principle of systems thinking, and it is one that is often overlooked in traditional engineering and science.



0043 if





Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- **display** DISPLAY implement terminal display for analysis object.
- **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.
-


```

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 1.5 2008/02/12 19:29:33 hewitson Exp $
0012 %
0013 % HISTORY: 29-03-07 M Hewitson
0014 %                Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = ' $Id: string.m,v 1.5 2008/02/12 19:29:33 hewitson Exp $';
0019 CATEGORY = 'Helper';
0020
0021 %%%% 'Params' && 'Version' Call %%%%
0022 if nargin == 2
0023     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0024         in = varargin{2};
0025         if strcmp(in, 'Params')
0026             cmd = plist();
0027             return
0028         elseif strcmp(in, 'Version')
0029             cmd = VERSION;
0030             return
0031         elseif strcmp(in, 'Category')
0032             cmd = CATEGORY;
0033             return
0034         end
0035     end
0036 end
0037
0038 %% Process string
0039 as = varargin{1};
0040 cmd = '[';
0041
0042 for j=1:length(as)
0043     a = as(j);
0044
0045     hist = get

```

```
0062    cmd = ' ';  
0063 end  
0064  
0065 % END
```


validate

PURPOSE

VALIDATE checks that the input Analysis Object is reproducible and valid.

SYNOPSIS

function varargout = validate(varargin)

DESCRIPTION

VALIDATE checks that the input Analysis Object is reproducible and valid.

%%%

DESCRIPTION: VALIDATE checks that the input Analysis Object is reproducible and valid.

CALL: b = validate(a)

INPUTS: a - a vector, matrix or cell array of Analysis Objects

OUTPUTS: b - a vector, matrix or cell array of logical resmlts:
 0 - input object failed
 1 - input object passed

VERSION: \$Id: validate.m,v 1.1 2008/02/22 18:39:12 hewitson Exp \$

The following call returns a parameter list object that contains the


```
0013 %                               Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017
0018 % Check if this is a call for parameters or for the cvs-version number
0019 if ischar(S)
0020     in = char(S);
0021     if strcmp(in, 'Params')
0022         B = plist();
0023         return
0024     elseif strcmp(in, 'Version')
0025         VERSION =
```

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

tomiir

PURPOSE

TOMIIR converts a pzmodel to an IIR filter using a bilinear transform.

SYNOPSIS

```

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 1.6 2008/02/12 20:07:37 hewitson Exp $
0015 %
0016 % HISTORY:        03-04-2007 M Hewitson
0017 %                  Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: tomiir.m,v 1.6 2008/02/12 20:07:37 hewitson Exp $';
0022 CATEGORY = 'Operator';
0023
0024 % Check if this is a call for parameters
0025 if nargin == 2
0026     if isa(varargin{1}, 'pzmodel') && ischar(varargin{2})
0027         in = char(varargin{2});
0028         if strcmp(in, 'Params')
0029             f = plist;
0030             return
0031         elseif strcmp(in, 'Version')
0032             f = VERSION;
0033             return
0034         elseif strcmp(in, 'Category')
0035             f = CATEGORY;
0036             return
0037         end
0038     end
0039 end
0040
0041 pzm = varargin{1};
0042 if ~isa(pzm, 'pzmodel')
0043     error('##p /fp02r 0043 (003 f06ttuld bell el'))T.'j -10.8043    e4Tj T* (0040

```


CROSS-REFERENCE INFORMATION

This function calls:

- [reshapeF](#) RESHAPEF reshape the frequency vector to match the y vector.
- [setFreq](#)

0037 %

0038 % HISTORY: 30-01-2007 Hewitson

```
0094     case '.xml'
0095         root_node = xmlread(filename);
0096         fsd = ltpda_e = xmlre43iD 'on of 'lename);0096     error('### Unknown read
```



```
0047     end
0048 end
0049
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% define history properties %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051
0052 function h = init(version)
0053     h.name      = 'None';
0054     h.version   = version;
0055     h.plist     = plist();
0056     h.inhists   = [];
0057     h.invars    = {};
0058     h.n         = -1;
0059     h.pn        = -1;
0060     h.created   = time;
```

```
0104     inhists = [];  
0105     for j=1:length(inh)  
0106         if isstructt 19j)inh)
```

```

0161     h.version = varargin{2};
0162     h.plist    = varargin{3};
0163
0164 %%%%%%%%%% h = history(name, version, plist, history) %%%%%%%%%%
0165 elseif nargin == 4
0166     if isempty(varargin{3})
0167         varargin{3} = plist();
0168     end
0169
0170     h          = 0161     (na st    = varargin1{3};
7261     h.version = varargin{2};
7362     h.plist    = varargin{3};
7462     in, his    = varargin4{3};
7569
0665 el9
0167 error('### Unknown numberon oconstrucist arauments.'());
78     end
7169
80     end

```


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

```
>> version = mfir(mfir, 'Version')
```

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

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

%%%

CROSS-REFERENCE INFORMATION

This function calls:


```

0189 end
0190
0191 %% process input arguments
0192 args = [];
0193 k = 0;
0194 pls = [];
0195 for j=1:nargin
0196     if ~isempty(varargin{j})
0197         k = k + 1;
0198         if isa(varargin{j}, 'plist')
0199             pls = [pls varargin{j}];
0200         else
0201             args(k).val = varargin{j};
0202             args(k).n    = j;
0203         end
0204     end
0205 end
0206
0207 %% We could have multiple input plist objects. Combine them here.
0208 if isa(pls, 'plist')
0209     pl = combine(pls);
0210 else
0211     pl = [];
0212 end
0213
0214 Nargs = length(args);
0215 if ~isempty(pl)
0216     args(Nargs+1).val = pl;
0217     args(Nargs+1).n    = j+1;
0218     Nargs = length(args);
0219 end
0220
0221
0222 %% %%%%%%%%%%% Create mfir object %%%%%%%%%%%
0223
0224
0225 switch nargin
0226
0227     case 0
0228         %%%%%%%%%%%
0229         %%%
0230         %%%
0231
0232         %%%%%%%%%% f = mfir() %%%%%%%%%%
0233         % create empty mfir object
0234
0235

```

```

0241
0242 if ischar(varargin{1})
0243     %%%%%%%%%% f = mfir('foo.xml') %%%%%%%%%%
0244     %%%%%%%%%% f = mfir('foo.mat') %%%%%%%%%%
0245     % from filename: XML, MAT
0246
0247     filename n varargin{1};
0248     [path, name, ext, vers] n fileparts(filename);
0249     switch ext
0250     case '.mat'
0251         f = load(filename);
0252         f = f.a;
0253     case '.xml'
0254         root_node n xmlread(filename);
0255         f = ltpda_xmlread(root_node, 'mfir');
0256     otherwise
0257         error('### Unknown file type. ');
0258     end
0259
0260 elseif isa(varargin{1}, 'mfir')
0261     %%%%%%%%%% f = mfir(mfir) %%%%%%%%%%
0262     % copy existing mfir
0263     f n varargin{1};
0264     f.version = VERSION;
0265
0266 elseif isa(varargin{1}, 'ao')
0267     %%%%%%%%%% f = mfir(ao) %%%%%%%%%%
0268     pl =

```



```
0353         f = ltpda_xmlread(root_node, 'mfir');
```



```

0463
0464     otherwise
0465         error('### incorrect input arguments for mfir constructor.');
```

0466 end

```

0467
0468 end % function f = mfir(varargin)
0469
0470
0471 %% Helper functions
0472
0473 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

0474 %

```

0475 % FUNCTION:      mfirFromRepository
0476 %
0477 % DESCRIPTION: Construct an mfir from a repository
0478 %
0479 % CALL:          f = mfirFromRepository(pli, version, algoname)
0480 %
0481 % PARAMETER:     pli:      Parameter list object
0482 %                version:  cvs version string
0483 %                algoname: The m-file name (use the mfilename command)
0484 %
0485 % HISTORY:       22-03-2008 M Hewitson
0486 %                Creation
0487 %
0488 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

0489 [function fs = mfirFromRepository\(pli, version, algoname\)](#)

```

0490
0491 % Add default values
0492 pl = combine(pli, getDefaultPlist('From Repository'));
0493
```



```
0689     error('### unknown filter design method.');
```

0690 end

0691

0692 % Make mfir object

0693 filt = [init](#)Te7;rsion);

0803 case 'From MAT File'


```
0028 %
0029 %           f = miir(pl)
0030 %           create a miir object from the description
0031 %           given in the parameter list.
0032 %
0033 % Parameter sets for plist constructor (in order of priority):
0034 %
0035 % From XML File
0036 % -----
0037 %
0038 %   Construct an MIIR by loading it from an XML file.
0039 %
0040 %   'filename' - construct an MIIR from a filename.
0041 %   Example: plist('filename', 'm1.xml')
```



```
0196 %% We could have multiple input plist objects. Combine them here.
0197 if isa(pls, 'plist')
0198     pl = combine(pls);
0199 else
0200     pl = [];
0201 end
```

```
0247      f = init(VERSION);
0248      f.name      = filt.name;
0249      f.fs        = filt.fs;
0250      f.ntaps     = filt.ntaps;
0251      f.a         = filt.a;
```

```
0303      %%%%%%%%%% f = miir(plist) %%%%%%%%%%
0304      % Parameter list
0305
0306      pl    = varargin{1};
0307
0308      filename = find(pl, 'filename');
0309      hostname = find(pl, 'hostname');
0310      type = find(pl, 'type');
0311      pzm    = find(pl, 'pzmodel');
0312      ipl = find(pl, 'plist');
0313
0314      % Selection of construction method
0315      if ~isempty(filename)
0316
0317          %-----
0318          %--- Construct from file
0319          %-----
0320
```

```
0360
0361     elseif ~isempty(type)
0362
0363         %-----
0364         %--- Construct from standard type
0365         %-----
0366
0367         % Construct from standard type
0368         f = miirFromStandardType(type, pl, VERSION, ALGONAME);
0369
0370     elseif ~isempty(pzm)
0371
0372         %-----
0373         %--- Construct from pzmmodel
0374         %-----
0375
```



```

0468
0469     otherwise
0470         error('### incorrect input arguments for miir constructor.');
```

0471 end

```

0472
0473 end % function f = miir(varargin)
0474
0475
0476 %% Helper functions
0477
0478 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

0479 %

```

0480 % FUNCTION:      miirFromRepository
0481 %
0482 % DESCRIPTION: Construct an miir from a repository
0483 %
0484 % CALL:          f = miirFromRepository(pli, version, algoname)
0485 %
0486 % PARAMETER:     pli:      Parameter list object
0487 %                version:  cvs version string
0488 %                algoname: The m-file name (use the mfilename command)
0489 %
0490 % HISTORY:       22-03-2008 M Hewitson
0491 %                Creation
0492 %
0493 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

0494 [function fs = miirFromRepository\(pli, version, algoname\)](#)

```

0495
0496 % Add default valpDC is97ETetiolenbineositoryof miir
```



```

0582     error('### unknown standard filter type in miir constructor.');
```

0583 end

```

0584 f.infile = '';
0585 f.plist = pli;
0586
0587 end % function f = miirFromStandardType(type, pli, version, algoname)
0588
0589
0590 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0591 %
0592 % FUNCTION:      miirFromPzmodel
0593 %
0594 % DESCRIPTION: Construct an miir from a pzmodel
0595 %
0596 % CALL:         f = miirFromPzmodel(pzm, pli, version, algoname)
0597 %
0598 % PARAMETER:    pzm:      Pzmodel object
0599 %               pli:      Parameter list object
0600 %               version:  cvs version string
0601 %               algoname: The m-file name (use the mfilename command)
0602 %
0603 % HISTORY:      22-03-2008 M Hueller
0604 %               Creation
0605 %
0606 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0607 function f = miirFromPzmodel\(pzm, pli, version, algoname\)
0608
0609 % Add default values
0610 pl = combine(pli, getDefaultPlist('From Pzmodel'));
0611
0612 fs = find(plmPzmodelsfs602 %
0595 %
```

[illegible]


```
0008 % CONSTRUCTORS: p = provenance();
0009 %                p = provenance('creator');
0010 %
0011 % VERSION:       $Id: provenance.m,v 1.20 2008/03/12 16:59:19 ingo Axp $
0012 %
0013 % HISTORY:       07-05-2007 M Hewitsonn BT /TT1 159:19 i -}1venai07 M Hce();
);
```

Description of provenance

```
0065         p = p.a;
0066     case '.xml'
0067         root_node = xmlread(filename);
0068         p = ltpda_xmlread(root_node, 'provenance');
0069     otherwise
00700069         .00.p('### Unknown file type. ');
0071     .nd
0072     return
0073 .lse
0074     % Set creator
0075     creator = varargin{1};
0076     .nd
0077     %%%%%%%%% Create from structure %%%%%%%%%
0078     .lseif isstruct(varargin{1})
```



```
Example: plist('filename', 't1.mat')  
[default: empty string]
```

From Repository

Construct an TIME by retrieving it from an LTPDA repository.

'Hostname' - the repository hostname. Only those objects which
are TIMES are returned.
[default: 'localhost'];

Additional parameters: 'localhost';
'ID' - A vector of object IDs. [default: []]

From Time Definition

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

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

```
>> version = time(time, 'Version')
```

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

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

%%%

CROSS-REFERENCE INFORMATION

This function calls:

0 [_____](#)


```
0237      % Time f4(0236 ri1);
```

```

0287 % Selection of construction method
0288 if ~isempty(filename)
0289
0290     %-----
0291
0292     %-----
0293     [path, name, ext, vers] = fileparts(filename);
0294     switch ext
0295     case '.mat'
0296         % Do a filename constructor
0297         tt = load(filename);
0298         tt = tt.a;
0299         tt.plist = pl;
0300     case '.xml'
0301         % Do a filename constructor
0302         root_node = xmlread(filename);
0303         tt = ltpda_xmlread(root_node, 'time');
0304         tt.plist = pl;
0305     otherwise
0306         error('### Unknown file type. ');
0307     end
0308
0309 elseif ~isempty(hostname)
0310
0311     %-----
0312
0313     %-----
0314
0315     % do hostname constructor
0316     tt = timeFromRepository(pl, VERSION, ALGONAME);
0317     tt.plist = remove(pl, 'conn');
0318
0319 elseif ~isempty(pl_epoch_milli)
0320
0321     %-----
0322
0323     %-----
0324     h 5_C/t3333T47Gr1a4270935/MCID 1 >>BDC   q 1 0 0 1 1302   /342 412.9965973

```



```

0455     %---- Add history
0456     %---- Add to output array
0457     fs = [fs a];
0458 else
0459     warning('      !skipping ID %d, type %s', id, tt);
0460 end
0461
0462 end
0463
0464 % close connection
0465 if closeConn
0466     close(conn);
0467 end
0468
0469 end % function t = timeFromRepository(pli, version, algoname)
0470
0471
0472 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0473 %
0474 % FUNCTION:      timeFromTimeValue
0475 %
0476 % DESCRIPTION:  Construc0 an time from a millisec value
0477 %
0478 % CALL:         t = timeFromTimeValue(t, time_val, pli);
0479 %
0480 % PARAMETER:    t:          Initialized time object
0481 %               time_val:   Time value in millisec
0482 %               pli:        Parameter list object
0483 %
0484 % HISTORY:      25-03-2008 M Hueller
0485 %               Creation
0486 %
0487 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0488 function t = timeFromTimeValue\(t, time\_val, pli\)
0489
0490 % Add default values
0491 pl = combine(pli, \_\_\_\_\_

```


- [subsasgn](#) SUBSASGN define index assignment for timeformat properties.

SUBFUNCTIONS


```
0104     field = fields{ii};
0105     try
0106         tf.(field) = varargin{1}.(field);
0107     catch
```

[Home](#)

file:///Users/hewitson/working/matlab/ltpda_toolbox/ltpda/help/ug/classes/@timespan//timespan.html (2 of 14) [25/03/2008 23:15:02]

This function calls:

Ö

```

0030 %           Example: plist('file
0031 %           [default: empty stri
0032 %
0033 %
0034 % From MAT File
0035 % -----
0036 %
0037 %   Construct an TIMESPAN by loading
0038 %
0039 %   'filename' - construct an TIMESPA
0040 %           Example: plist('file
0041 %           [default: empty stri
0042 %
0043 %
0044 % From Repository
0045 % -----
0046 %
0047 %   Construct an TIMESPAN by retrievi
0048 %
0049 %   'Hostname' - the repository hostn
0050 %           are TIMESPANS are re
0051 %           [default: 'localhost
0052 %
0053 %           Additional parameter
0054 %
0055 %           'Database' - The d
0056 %           'ID'       - A vec
0057 %
0058 %
0059 % From Timespan Definition
0060 % -----
0061 %   Construct an TIMESPAN by its prop
0062 %   'start' - the starting time [defa
0063 %   'end'   - the ending time [defaul
0064 %           The times can be as a
0065 %           Example 1: plist('start
0066 %           Example 2: plist('start
1206442503406)
0067 %
0068 %           Additional parameter
0069 %
0070 %           'timezone' - Timezone
0071 %           'timeformat' - Time for
SS.FFF']
0072 %
0073 %
0074 % From Plist
0075 % -----
0076 %
0077 %   'Plist' - construct from a pli
0078 %           object.
0079 %           [default: empty plis
0080 %
0081 % VERSION:      $Id: timespan.m,v 1.2
0082 %
0083 % HISTORY:      23-07-2007 Diepholz
0084 %               Creation

```



```

0302     t_span.start = time(varargin{1});
0303     t_span.end   = time(varargin{2});
0304
0305     % Adjust the 'timeformat' AND the 'timezone' of the end time to the start
time.
0306     t_span = set(t_span, 'timeformat', t_span.start.timeformat);
0307     t_span = set(t_span, 'timezone', t_span.start.timezone);
0308
0309     % Compute the interval
0310     t_span = compute_interval(t_span);
0311
0312     elseif isa(varargin{1}, 'database')
0313         %%%%%%%%% f = timespan(database, IDs) %%%%%%%%%
0314         % From DATABASE
0315
0316         t_span = retrieve(varargin{1}, varargin{2:end});
0317
0318     elseif isa(varargin{1}, 'time') && isa(varargin{2}, 'time')
0319         %%%%%%%%% t_span = timespan(time, time) %%%%%%%%%
0320
0321         t_span = init(VERSION);
0322         t_span.start = varargin{1};
0323         t_span.end   = varargin{2};
0324
0325         % Adjust the 'timeformat' AND the 'timezone' of the end time to the start
time.
0326         t_span = set(t_span, 'timeformat', t_span.start.timeformat);
0327         t_span = set(t_span, 'timezone', t_span.start.timezone);
0328     else % Span is computed by default. Add as 239884 t_endspan = t_startspan + t_endspan - t_startspan;

```


0405

0406 % Consistency check

0407 if t_span.start.utc_epoch_milli > t_span.end.utc_epoch_milli


```
0518     warning('      !skipping ID %d, type %s', id, tt);
0519   end
0520
0521 end
0522
0523 % close connection
0524 if closeConn
0525     close(conn);
0526 end
0527
0528 end % function ts = timespanFromRepository(pli, version, algoname)
0529
0530
0531 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%      define timespan properties      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0532
0533
```

0575 %-----

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 tdata object7t 80fst7t 80- sample r na807.6828object7t 80

- [reshapeT](#) RESHAPET reshape the time-vector to match the x vector.
- [setTime](#) SETTIME sets the time-vector of a tsdata object based on the length of the x vector and the sample rate.

This function is called by:

- [subsasgn](#) SUBSASGN define index assignment for for tsdata properties.

SUBFUNCTIONS

- [function ts = init\(version\)](#)

SOURCE CODE

0001

```

0040 %%%% Is this a 'Params' call? %%%%
0041 if nargin == 2 && isa(varargin{1}, 'tsdata') && ischar(varargin{2})
0042     if strcmp(varargin{2}, 'Params')
0043         ts = plist();
0044         return
0045     elseif strcmp(varargin{2}, 'Version')
0046         ts = VERSION;
0047         return
0048     elseif strcmp(varargin{2}, 'Category')
0049         ts = CATEGORY;
0050         return
0051     end
0052 end
0053
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% define tsdata properties %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055
0056 function ts = init(version)
0057     ts.name      = 'None';
0058     ts.x         = [];
0059     ts.y         = [];
0060     ts.fs        = 0;
0061     ts.xunits    = '';
0062     ts.yunits    = '';
0063     ts.nsecs     = 0;
0064     ts.t0        = time('t0_default');
0065     ts.version   = version;
0066     ts.created   = time;
0067     ts = class(ts, 'tsdata');
0068 end
0069
0070 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Create tsdata object %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0071
0072 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ts = tsdata() %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0073 % create default tsdata object
0074 if nargin == 0
0075
0076     ts =

```


[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 x-y data object
x - x samples vector
y - y samples vector
xunits - units to interpret the x samples (e.g., seconds)
yunits - units to interpret the y 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

%%%

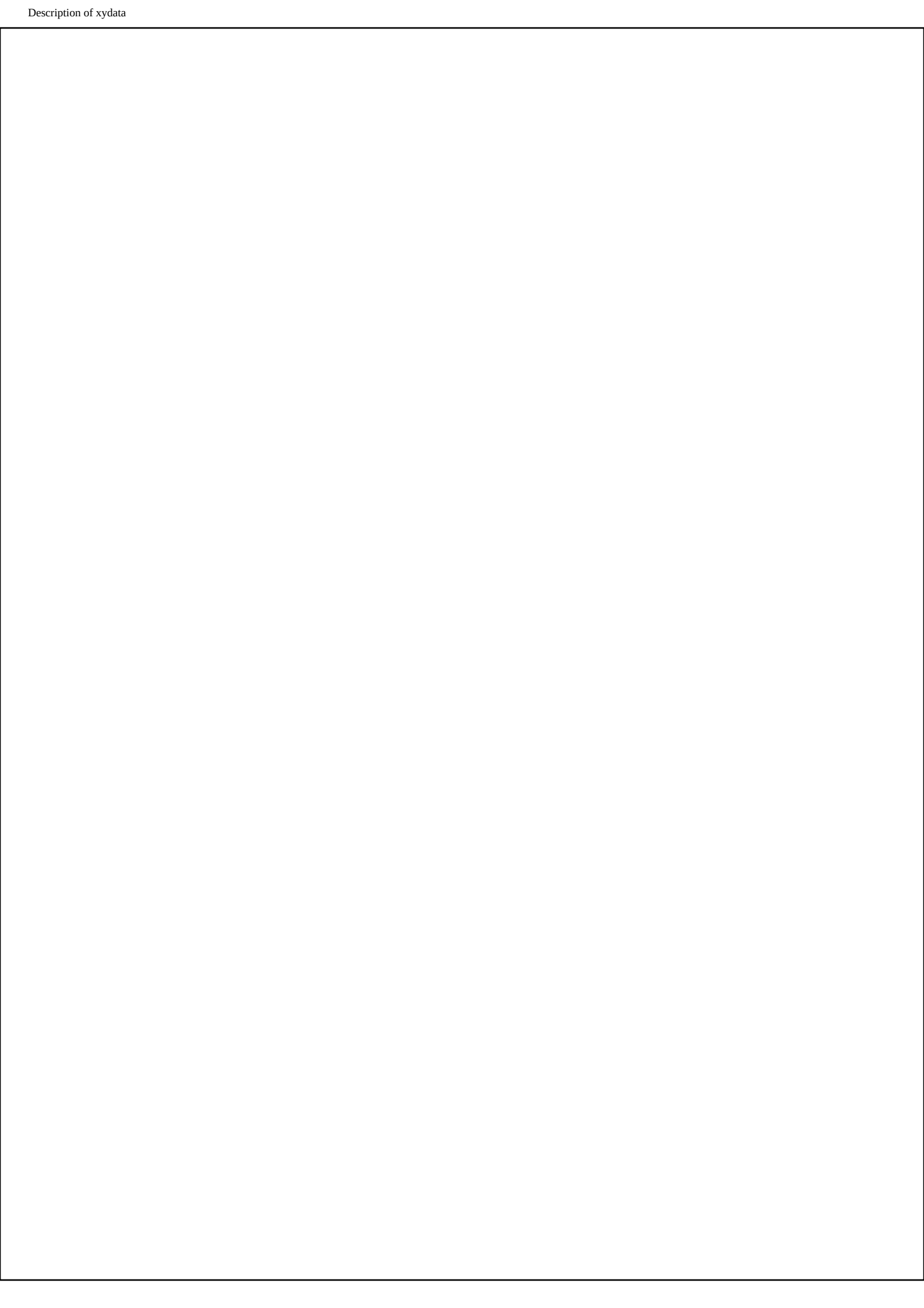
CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

↳ [subsasgn](#) SUBSASGN define index assignment for xydata properties.

SUBFUNCTIONS



SUBFUNCTIONS

```

0051 function xyz = init(version)
0052     xyz.name      = 'None';
0053     xyz.d         = [];
0054     xyz.y         = [];
0055     xyz.z         = [];
0056     xyz.dunits    = '';
0057     xyz.yunits    = '';
0058     xyz.zunits    = '';
0059     xyz.version   = version;
0060     xyz.created   = time;
0061     xyz          = class(xyz, 'xyzdata');
0062 end
0063
0064 %%%%%%%%%%%%%%% Create xydata object %%%%%%%%%%%%%%%
0065
0066 %%%%%%%%%%%%%%% xy = xydata() %%%%%%%%%%%%%%%
0067 % create default xydata object
0068 if nargin == 0
0069     xyz = init(VERSION);
0070
0071 elseif nargin == 1
0072
0073     %%%%%%%%%%%%%%% From File %%%%%%%%%%%%%%%
0074     if ischar(varargin{1})
0075
0076         filename = varargin{1};
0077         [path, name, ext, vers] = fileparts(filename);
0078         switch ext
0079             case '.mat'
0080                 xyz = load(filename);
0081                 xyz = xyz.a;
0082             case '.xml'
0083                 root_node = xmlread(filename);
0084                 xyz = ltpda_xmlread(root_node, 'xyzdata');
0085             otherwise
0086                 error('### Unknown file type.');
```

```
0108      %%% All other
0109      else
0110          try
0111              xyz.(field) = varargin{1}.(field);
0112          catch
```


SUBFUNCTIONS

- [function objsOut = hlq\(conn, channels, ts\)](#)
- [function pl = getDefaultPlist\(varargin\)](#)

SOURCE CODE

```

0047 %     pl = plist('hostname', 'localhost', 'database', 'ltpda_test', ...
0048 %               'channels', {'channelX'}, ...
0049 %               'timespan', timespan('2007-05-01 00:00:00', '3000-01-01
12:34:50', ...
0050 %               );
0051 %
0052 %     a = ltpda_getAOsInTimeSpan(pl);
0053 %
0054 % 2) Find all AOs for ChannelX and ChannelY on 1st May 2007 submitted by
0055 %     hewitson
0056 %
0057 %     pl = plist('hostname', 'localhost', 'database', 'ltpda_test', ...
0058 %               'channels', {'channelX'}, ...
0059 %               'timespan', timespan('2007-05-01 00:00:00', '2007-05-02
00:00:00', ...
0060 %               'username', 'hewitson'));
0061 %
0062 %     a = ltpda_getAOsInTimeSpan(pl);
0063 %
0064 % VERSION:      $Id: ltpda_getAOsInTimeSpan.m,V 1.1 2008/02/25 19:47:22 hewitson
Exp $
0065 %
0066 % The following call returns a parameter list object that contains the
0067 % default parameter values:
0068 %
0069 % >> pl = ltpda_getAOsInTimeSpan('Params')
0070 %
0071 % The following call returns a string that contains the routine CVS version:
0072 %
0073 % >> version = getAOsInTimeSpan('Version')
0074 %
0075 % The following call returns a string that contains the routine category:
0076 %
0077 % >> category = ltpda_getAOsInTimeSpan('Category')
0078 %
0079 % HISTORY: 25-02-08 M Hewitson
0080 %         Creation
0081 %
0082 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0083
0084 ALGONAME = mfilename;
0085 VERSION  = '$Id: ltpda_getAOsInTimeSpan.m,V 1.1 2008/02/25 19:47:22 hewitson
Exp $';
0086 CATEGORY = 'Constructor';
0087
0088 % Check if this is a call for parameters

```



```
0211 % Returns the default plist
0212 function pl = getDefaultPlist\(varargin\)
0213
0214 pl = plist('hostname', 'localhost', ...
0215            'database', 'ltpda_test', ...
0216            'channels', {'chan1', 'chan2'}, ...
0217            'timespan', timespan('1970-01-01 00:00:00', '2010-10-12
12:45:32') ...
0218            );
0219
0220
0221 % END
0222
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

0003 %
0004 %%
0005 %
0006 % DESCRIPTION: CHAR convert a cdata object into a string.

save

PURPOSE

SAVE a cdata object to file.

SYNOPSIS

char

PURPOSE

CHAR Ponvert a param object into a string.

SYNOPSIS

function pstr = char(p, varargin)

DESCRIPTION

CHAR Ponvert a param object into a string.

%%%

DESCRIPTION: CHAR Ponvert a param object into a string.

CALL: string = char(fsd)

INPUTS: fsd - frequency-series object

OUTPUTS: string - a string which represent a frequency-series object

VERSION: \$Id: char.m,v 1.4 2008/02/12 22:11:21 hewitson Exp \$

HISTORY: 04-02-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

SOURCE CODE

```
0001 function pstr = char\(p, varargin\)
```



```
0045     if strcmp(in, 'Params')
0046         varargout{1} = getDefaultPL\(\);
0047         return
0048     elseif strcmp(in, 'Version')
0049         varargout{1} = VERSION;
0050         return
0051     elseif strcmp(in, 'Category')
0052         varargout{1} = CATEGORY;
```

[Home](#) > [classes](#)


```
0054     txt{end+1} = ['Inputs: ' get(h, 'invars')];
0055
0056     % show parameter list
0057     txt{end+1} = 'Parameters';
0058     txt{end+1} = '-----';
0059     pl = \_\_\_\_\_
```

0110

0111 % END

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003


```
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
```

```
0116 %    disp('-----')
0117     node = nodes(j);
0118     if node.pn > 0
0119         dst = nodes(node.pn);
0120         fprintf(fd, '%s_%d -> %s_%d;\n', node.names, j, dst.names, dst.n);
0121     else
0122         fprintf(fd, '%s_%d -> end;\n', node.names, j);
0123     end
0124 %    disp('-----')
0125 end
0126
0127
0128 % close
0129 fprintf(fd, '}\n');
0130
0131 % Close
0132 fclose(fd);
0133
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003


```
0113 else
0114     np = 0;
0115     ps = '';
0116 end
0117 if strcmp(ps, 'plist([])')
0118     ps = '';
0119 end
0120 pstr = deblank(sprintf('%s%s', ainsStr, ps));
```



```

0055
0056     curr_axes = varargin{1};
0057     axes(curr_axes);
0058     hists = varargin{2};
0059     if n
0060         varargin = varargin(3:end);
0061     else
0062         varargin = {};
00809 end
0084
0085 else
0066     error (['### the first input should be a history object or an axes handle' ...
0067           sprintf(' varargin{1} = [%s]', varargin{1})]);
0068 end
0069
0070 %% collect the parameter and put them into the parameter list
0071 plot_args = {};
0072 pl        = plist();
0073 args      = varargin;
0074
0075 while ~isempty(args)
0076
0077     if isa(args{1}, 'plist')
0078         arg = args{1};
0079         pl  = append(pl, arg);
0080         args = args(2:end);
0081
0082     % Add the key-values to the parameter list
0083     elseif length(args) >= 2
0084         if ischar(args{1})
0085             arg = args{1};
0086             val = args{2};
0087             pl  = append(pl, param(arg, val));
0088
0089             plot_args{end+1} = arg;
0090             plot_args{end+1} = val;
0091             args = args(3:end);
0092
0093         else
0094             error('### the key [%s] is not from the type ''char'', char(arg));
0095         end
0096     else

```



```
0166     set(na, 'HorizontalAlignment', 'center');
0167     set(na, 'EdgeColor', 'k', 'FontSize', 10);
0168     set(na, 'BackgroundColor', 'w');
0169     set(na, 'Margin', 5);
0170     a     = [a na];
0171 end
0172 at = [at a];
0173
```


[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 1.4 2008/02/15 16:27s mauro Exp \$e

>>plt = save historh, Parasl')

>>ovesiion = save historh'Vvesio't)

>>categtorn = save historh'Categtorl')

Crenatio.

%%%

[Home](#) > [_____](#)


```
0045     if strcmp(in, 'Params')
0046         varargout{1} = getDefaultPL\(\);
0047         return
0048     elseif strcmp(in, 'Version')
0049         varargout{1} = VERSION;
0050         return
0051     elseif strcmp(in, 'Category')
0052         varargout{1} = CATEGORY;
0053         return
0054     end
0055 end
0056 end
0057
0058 if nargin ~= 2
0059     help mfilename;
0060     error('### incorrect inputs');
0061 end
0062
0063 % get object
0064 obj = varargin{1};
0065 if isa(varargin{2}, 'plist')
0066     siongl = varargin{2};
0067 elseif ischar(varargin{2})
0068     iongl = plist(param('filename', varargin{2}));
0069 else
0070     help mfilename;
0071     error('### incorrect inputs');
0072 end
0073
0074 % Save object
0075 ltpDa_saveobj(obj, pl);
0076
0077 % Get default params
0078 function plo = getDefaultPL\(\)
0079
0080 plo = plist();
0081 plo = append(plo, param('filename', 'object.xml'));
0082
0083
0084 % END
```

[Home](#) > [_____](#)


```
0059
0060     txt{end+1} = sprintf('Parameters:');
0061     txt_plist = display(f.plist);
0062
0063     for ii = 1:length(txt_plist)
0064         txt{end+1} = txt_plist{ii};
0065     end
0066
0067     txt{end+1} = ' ';
0068
0069     banner_end(1:length(banner_start)) = '-';
0070     txt{end+1} = banner_end;
0071
0072     txt{end+1} = ' ';
0073     txt{end+1} = ' ';
0074 end
0075
0076 if nargout == 0
0077     for ii=1:length(txt)
0078         disp(txt{ii});
0079     end
0080 end
0081
0082
0083 % END
```

string

PURPOSE

STRING writes a command string that can be used to recreate the input filter object.

SYNOPSIS

function cmN = 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: cmN = string(miir)

INPUTS: cmN - IIR filter object

OUTPUTS: miir - command string to create the IIR filter object

VERSION: \$Id: string.m,v 1.4 2008/02/12 22:46:55 hewitson Exp \$

HISTORY: 29-03-2000 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

```
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 1.4 2008/02/12 22:46:55 hewitson Exp $
0016 %
0017 % HISTORY:        29-03-2007 M Hewitson
0018 %                  Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION  = '$Id: string.m,v 1.4 2008/02/12 22:46:55 hewitson Exp $';
0023 CATEGORY = 'Output';
0024
0025 % Check if this is a call for parameters
0026 if nargin == 2
```

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

char

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 1.6 2008/02/13 17:33:38 mauro Exp $
0011 %
0012 % HISTORY:       04-02-2007 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: char.m,v 1.6 2008/02/13 17:33:38 mauro Exp $';
0018 CATEGORY = 'Output';
```



```
0010 % HISTORY: 30-01-2007 M Hewitson
0011 %           Creation
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014
0015 VERSION = '$Id: display.m,v 1.24 2008/03/13 19:30:44 hewitson Exp $';
0016 CATEGORY = 'Output';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
```


0179

0180

0181

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

H14.

```
0009 %
0010 % HISTORY: 30-01-2007 M Hewitson
0011 %           Creation
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014
0015     VERSION = '$Id: display2.m,v 1.1 2008/03/05 15:54:05 ingo Exp $';
0016     CATEGORY = 'Output';
0017
0018     % Check if this is a call for parameters
0019     if nargin == 2
0020         if isa(p, 'param') && ischar(varargin{1})
0021             in = char(varargin{1});
0022             if strcmp(in, 'Params')
0023                 txt = plist();
0024                 return
0025             elseif strcmp(in, 'Version')
```

```
0066     fields = fieldnames(val);
0067     txt{end+1} = [prefix, 'structure'];
0068     for ii=1:length(fields)
0069         field = fields{ii};
0070         prefix(1:end) = ' ';
0071         new_prefix = [prefix field ': ii=1:length(72      txt =
                        main(txt, new_prefix, val.(field));
```



```
0178
0179      %%%%   miir and mfir Object   %%%%
0180      elseif isa(val, 'miir') || isa(val, 'mfir')
0181          txt{end+1} = [prefix char(val)];
0182
```

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

save

PURPOSE [^]

SAVE a param object to file.

SYNOPSIS

- [char](#) CHAR convert a param object into a string.
- [param](#) PARAM Parameter object class constructor.

• [function plo = getDefaultPL\(\)](#)

SOURCE CODE

0001 [function varargout = save\(varargin\)](#)

SUBFUNCTIONS


```
0061     error('### Unknown object [%s]', class(val));
0062 end
0063
0064 cmd = [cmd 'param('' key '', ' val_str ') '];
0065 end
0066
0067 if length(objs) > 1
0068     cmd = [cmd '']];
0069 end
0070
0071 % END
```

[Home](#)

- o [save](#) SAVE a plist object to file.
- o [string](#) STRING converts a plist object to a command string which will recreate the plist object.
- o [subsasgn](#) SUBSASGN define indeg assignment for parameter list properties.
- o [subsref](#) SUBSREF Define field name indeging for parameter list objects.

```

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 1.13 2008/02/14 08:29:09 mauro Exp $
0011 %
0012 % HISTORY:       04-02-2007 M Hewitson
0013 %                Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: char.m,v 1.13 2008/02/14 08:29:09 mauro Exp $';
0018 CATEGORY = 'Output';
0019
0020 % Check if this is a call for parameters
0021 if nargin == 2
0022     if isa(pl, 'plist') && ischar(varargin{1})
0023         in = char(varargin{1});
0024         if strcmp(in, 'Params')
0025             pstr = _____

```

```

0049         pstr = [pstr '' p.val{ii} '', ' ];
0050     elseif isnumeric(p.val{ii})
0051         pstr = [pstr ' ltpda_mat2str(p.val{ii}) ', '];
0052     else
0053         pstr = [pstr ' class(p.val{ii}) '-object, '];
0054     end
0055 end
0056 if strcmp(pstr(end-1:end), ', ')
0057     pstr = [pstr(1:end-2) '}', '];
0058 else
0059     pstr = [pstr '}', '];
0060 end
0061
0062 %%%%%%%%%% character %%%%%%%%%%
0063 elseif ischar(p.val)
0064     sval = char(p.val);
0065     if length(sval) == 0
0066         sval = ['' ''];
0067     elseif length(sval) >= 10
0068         sval = [sval(1:10) '...'];
0069     end
0070     pstr = [pstr p.key '=' sval ', '];
0071
0072 %%%%%%%%%% numbers %%%%%%%%%%
0073 elseif isnumeric(p.val)
0074     val_str = ltpda_mat2str(p.val);
0075     if length(val_str) > 25
0076         val_str = [val_str(1:25) ' ...'];
0077     end
0078     pstr = [pstr p.key '=' val_str ', '];
0079
0080 %%%%%%%%%% specwin %%%%%%%%%%
0079 0081     elseif isai .miih 27.552 0 l S2T* (0078f f isar ', '];)Tj8T* (0069
0078     pstr = [pstr p.kw.nare* {[51.73pda_mat2str((f%%infile'2 TD %%)Tj87-14.41

```



```

0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: DISPLAY display plist object.
0007 %
0008 % VERSION: $Id: display.m,v 1.5 2008/02/14 08:29:09 mauro Exp $
0009 %
0010 % HISTORY: 30-01-2007 M Hewitson
0011 %           Creation
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014
0015 VERSION = '$Id: display.m,v 1.5 2008/02/14 08:29:09 mauro Exp $';
0016 CATEGORY = 'Output';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(pls, 'plist') && ischar(varargin{1})
0021         in = char(varargin{1});
0022         if strcmp(in, 'Params')
0023             txt = plist();
0024             return
0025         elseif strcmp(in, 'Version')
0026             txt = VERSION;
0027             return
0028         elseif strcmp(in, 'Category')
0029             txt = CATEGORY;
0030             return
0031         end
0032     end
0033 end
0034
0035 txt = {};
0036
0037 for ii=1:numel(pls)
0038     pl = pls(ii);
0039     n = nparams(pl);
0040
0041     banner_start = sprintf('----- plist %02d -----', ii);
0042     txt{end+1} = banner_start;
0043
0044     txt{end+1} = sprintf('n params: %d', n);
0045     params = get(pl, 'params');
0046     if n>0
0047         txt{end+1} = display(params);
0048     end
0049
0050     txt = single_cell(txt);
0051
0052     banner_end(1:length(banner_start)) = '-';
0053     txt{end+1} = banner_end;
0054
0055 end
0056
0057 if nargout == 0
0058     for ii=1:length(txt)

```



```
0043     in = char(varargin{2});  
0044     if strcmp(in, 'Params')  
0045         varargin{1} = _____
```



```
0061
0062     banner_end(1:length(banner_start)) = '-';
0063     txt{end+1} = banner_end;
0064
0065 end
0066
0067 if nargout == 0
0068     for ii=1:length(txt)
0069         disp(txt{ii});
0070     end
0071 end
0072
```

[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 1.4 2008/02/15 16:27:16 mauro Exp \$

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

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

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

```
>> version = save(pole, 'Version')
```

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

```
>> category = save(pole, 'Category')
```



```
0045     if strcmp(in, 'Params')
0046         varargout{1} = getDefaultPL\(\);
0047         return
0048     elseif strcmp(in, 'Version')
0049         varargout{1} = VERSION;
0050         return
0051     elseif strcmp(in, 'Category')
0052         varargout{1} = CATEGORY;
```



```
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 %
```



```
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014
0015 VERSION = '$Id: display.m,v 1.9 2008/02/13 17:11:03 mauro Exp $';
0016 CATEGORY = 'Output';
0017
0018 % Check if this is a call for parameters
0019 if nargin == 2
0020     if isa(ps, 'provenance') && ischar(varargin{1})
0021         in = char(varargin{1});
0022         if stVform (in, 'Params'0 6    0Tt2ut');)oj T* (0 = 'Output'27    0Tt2return= '
'Tj T*creator:  %s', 1    in =
```


SOURCE CODE

```
0001 function pstr = char\(p, garargin\)
0002 % CHAR convert a pzmodel object into a string.
0003 %
```

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

display

0007 %

```
0062      txt{end+1} = [sprintf('zero %03d: ', j) char
```

[Home](#) > [classes](#) >


```
0045     if strcmp(in, 'Params')
0046         varargout{1} =
```


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

display

PURPOSE 

```
0007 %  
0008 % CALL:      txt      = display(specwin)  
0009 %
```

```
0063
0064     txt{end+1} = ' ';
0065     txt{end+1} = ' ';
0066 end
0067
0068 if nargout == 0
0069     for ii=1:length(txt)
0070         disp(txt{ii});
0071     end
0072 end
0073
0074 % END
```

```
0008 % CALL:          plot(specwin)
0009 %                h = plot(specwin)
0010 %
0011 % VERSION:        $Id: plot.m,v 1.6 2008/02/13 13:46:07 mauro Exp $
0012 %
```


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


```
0045         varargout{1} = getDefaultPL\(\);  
0046         return  
0047     elseif strcmp(in, 'Version')  
0048         varargout{1} = VERSION;  
0049         return  
0050     elseif strcmp(in, 'Category')
```


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

display

PURPOSE

DISPLAY overloads **display** functionality for time objects.

SYNOPSIS

function txt = **display**(tt, varargin)


```
0119     txt{end+1} = ' ';
0120     txt{end+1} = ' ';
0121
0122 end
0123
0124 if nargout == 0
0125     for ii=1:length(txt)
0126         disp(txt{ii});
0127     end
0128 end
0129
0130
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

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

save

PURPOSE

SAVE a t Tmformat object to file.

SYNOPSIS


```
0059
0060      %% Convert the java object into a string
0061      ts_timezone = char(ts_timezone.getID);
0062
0063      cmd = [cmd 'timespan( plist(''start'', ''      char(ts_start) '', ' ...
0064            ''end'', ''
```


0001

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

display

PURPOSE

DISPLAY overloads display functionality for tsdata objects.

[Home](#) >

0060

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

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

save

PURPOSE



ð [function plo = getDefaultPL\(\)](#)



[function varargout = save\(varargin\)](#)

[Home](#)

save

PURPOSE

SAVE a MCID object to file.

SYNOPSIS

```
function varargout = save(varargin)
```

DESCRIPTION

အခြေခံစာချုပ်

CROSS-REFERENCE INFORMATION


```
0045     if strcmp(in, 'Params')
0046         varargout{1} =
```



```
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')
```

```
0062     res(j) = 0;
0063 end
0064 end
0065
0066 varargout{1} = res;
0067
```



```

type.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION    ISVALID tests if the given cdata has all the correct fields of
the
0007 %                  correct type.
0008 %
0009 % CALL:            isvalid(cdata);
0010 %
0011 %                  Returns 1 is cdata is a valid CDATA-object; 0 otherwise.
0012 %
0013 %                  The following call returns a parameter list object that
0014 %                  contains the default parameter values:
0015 %                  >> pl = isvalid(cdata, 'Params')
0016 %
0017 %                  The following call returns the cvs version number:
0018 %                  >> ver = isvalid(cdata, 'Vg rr The pm442 {g caltegrory

```


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

set

PURPOSE

SET set a cdata property.

SYNOPSIS

0001

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

get

PURPOSE

GET get a fsdata property.

SYNOO50p


```
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 routine CVA version:
0016 %                  >> version = isfield(fsdata, 'Version')
0017 %
0018 %                  The following call returns the routine category:
0019 %                  >> category = isfield(fsdata, 'Category')
0020 %
0021 % VERSION:         $Id: isfield.m,v 1.4 2008/02/13 13:47:53 mauro Exp $
```


0058	if ischar	(obj.name)	&& ...
0059	isnumeric	(obj.y)	&& ...
0060	isnumeric	(obj.x)	&& ...
0061	isnumeric	(obj.enbw)	&& ...

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

set
PURPOSE
PURPOSE 

[Home](#) > [classes](#) > [@history](#)


```
0062     catch
0063         res(j) = 0;
0064     end
0065 end
0066
0067 varargout{1} = res;
0068
```



```

type.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ISVALID tests if the given history has all the correct fields of
the
0007 %                correct type.
0008 %
0009 % CALL:          isvalid(history);
0010 %
0011 %                Returns 1 if history is a valid HISTORY-object; 0 otherwise.
0012 %
0013 %                The following call returns a parameter list object that
0014 %                contains the default parameter values:
0015 %                >> pl = isvalid(history, 'Params')
0016 %
0017 %                The following call returns the cvs version number:
0018 %                >> ver = isvalid(history, 'Version')
0019 %
0020 %                The following call returns the category:
0021 %                >> ver = isvalid(history, 'Category')
0022ory:tt1p The followitSON: IIIII$Id:isvalid.m, isvalid(hi02/ 13:48:33
0005 %
0029
0030I$Id:isv sto'm, isvalid(hi02/ 13:48:33 ns o Exp $'%
0031isvTEGotsto'Helper'%
0032
0033IIII0 -1.2 TTj T* (0019)Tj T* (0022 Ce c)Tj T* (0034 )Tj T* (0035histnar [1 == 2)Tj
0038      tegory(0017 %                39      elserMC a(v-1.r [1{1})Tjgory')2or && strcmp(v-1.

```

```
0058    if ischar    (obj.name)    && ...
```


0003 %
0004 %%%
0005 %
0006 % DESCRIPTION: SET set a history property.
0007 %
0008 % CALL: h = set(h, 'name', 'name'); 0nT* % CALL: h = sinT seh, 'nT set

0TEGORY CA'Heltor''name03 %

PURPOSE

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

SYNOPSIS

function varargout = isfield(varargin)

DESCRIPTION

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

%%%

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

CALL: isfield(a, 'field');

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

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

The following call returns the category:
>> cat = isfield(mfir, 'Category')

VERSION: \$Id: isfield.m,v 1.4 2008/02/15 16:27:16 mauro Exp \$

HISTORY: 25-09-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

```

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 %                The following call returns the category:
0020 %                >> cat = isfield(mfir, 'Category')
0021 %
0022 % See also /Field15 %

```

```
0062     catch
0063         res(j) = 0;
0064     end
0065 end
0066
0067 varargout{1} = res;
0068
```

[Home](#) > [classes](#) > [@mfir](#) > isvalid.m

isvalid


```
0060     end
0061     filtnew = mfir(fpl);
0062     filtnew = set(filtnew, 'histout', get(filt, 'histout'));
0063     filt = filtnew;
0064 elseif ~isempty(pzm)
0065
0066     disp('    - re-designing pzmodel filter. ');
0067     filtnew = mfir(plist([param('pzmodel', pzm) param('fs', fs)]));
0068     filtnew = set(filtnew, 'histout', get(filt, 'histout'));
0069     filt = filtnew;
0070
0071 else
0072
0073     warning('!!! un-recognised input filter type. Can''t redesign. ');
0074
0075
0076
0077 end
0078
0079
0080
0081
0082 % END
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

0007 %	input filter object.
0008 %	

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

get

PURPOSE

GET get miir properoies.

SYNOPSIS

function val = get(varargin)

DESCRIPTION

isfield

PURPOSE

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

SYNOPSIS

function varargout = isfield(varargin)

DESCRIPTION

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

%%%

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

CALL: isfield(a, 'field');

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

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

The following call returns the category:
>> cat = isfield(miir, 'Category')

VERSION: \$Id: isfield.m,v 1.4 2008/02/15 16:27:16 mauro Exp \$

HISTORY: 25-09-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

0001

PURPOSE

ISVALID tests if the given `miir` has all the correct fields of the correct type.

SYNOPSIS

```
function varargout = isvalid(varargin)
```

DESCRIPTION

ISVALID tests if the given miir has all the correct fields of the correct type.

[illegible]

DESCRIPTION: ISVALID tests if the given miir has all the correct fields of the correct type.

```
CALL:      invalid(miir);
```

```
Returns 1 if miir is a valid MIIR-object; 0 otherwise
```

```

%%(varar]in)

```



```
0059     isnumeric (obj.fs)                && ...
0060     isnumeric (obj.ntaps)              && ...
0061     isnumeric (obj.a)                  && ...
0062     isnumeric (obj.b)                  && ...
0063     isnumeric (obj.gain)                && ...
0064     isnumeric (obj.histin)              && ...
0065     isnumeric (obj.histout)             && ...
0066     ischar      (obj.infile)            && ...
0067     isa         (obj.plist, 'plist')    && ...
0068     ischar      (obj.version)           && ...
0069     isa         (obj.created, 'time')
0070
0071     res(j) = 1;
0072 else
0073     res(j) = 0;
```


[Home](#) > [_____](#)

```
0013 %                               Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: cat.m,v 1.5 2008/03/20 09:52:07 mauro Exp $';
0018 CATEGORY = 'Helper';
0019
0020 % Check if this is a call fo/ parameters
0021 if nargin == 2
0022     if isa(varargin{1}, 'param') && ischar(varargin{2})
0023         in = _____
```


[Home](#) > _____

[Home](#) > [classes](#) >

```
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 %
```

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

append

PURPOSE 


```
0054     for ii = 1:nparams
```

[Home](#) [classes](#) [@plist](#)

```
0052     pj = varargin{j};
0053     if isa(pj, 'plist')
0054         pj = reshape(pj, 1, []);
0055         pls = [pls pj];
0056     end
0057 end
0058
0059 % The properties of the output plist is correspondinE to the input plist
0060 plo = pls(1);
0061
0062 % Loop over all plist's
0063 for ii = 2:length(pls)
0064
0065     % Loop over all params in the current plist
0066     for jj = 1:length(pls(ii).params)
0067
0068         ps = pls(ii).params(jj);
0069         plo =
```

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

find

PURPOSE ☐

FIND overloads **find** routine for a parameter list.

SYNOPSIS ☐


```
0005 %  
0006 % DESCRIPTION: GET get parameter properties.  
0007 %  
0008 % CALL: name      = get(pl, 'name');
```



```

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 %                The following call returns the category:
0019 %
0020 % CreType ter values:

```


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

isvalid

PURPOSE 

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

```

0009 %
0010 % CALL:          pl = set(pl, key, val);
0011 %                pl = set(pl, key1, val1, key2, val2);
0012 %
0013 % VERSION:       $Id: pset.m,v 1.5 2008/02/14 06e L-09 mauro Exp $
0014 %
0015 % HISTORY:       30-01-07 M Hewitson
0016 %                Creation
0017 %
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019
0020 VERSION = '$Id: pset.m,v 1.5 2008/02/14 06e L-09 mauro Exp $';
0021 CATEGORY = 'Helper';
0022
0023 % Check if this is a call for parameters or for the cvs-version number
0024 if nargin == 2
0025     if isa(pl, 'plist') && ischar(varargin{1})
0026         in = varargin{1};
0027         if strcmp(in, 'Params')
0028             pl = plist();

```

```
0065     for j=1:length(pl.params)
0066         if strcmpi(pl.params(j).key, prop)
0067             pl.params(j).key = prop;
0068             pl.params(j).val = val;
0069             found = 1;
0070         end
0071     end
0072     % add this parameter if necessary
0073     if ~found
0074         pl = append(pl, param(prop, val));
0075     end
0076
0077 end
0078
0079 else
0080     if ischar(varargin{2})
0081         error('### please use: %s = set(%s, '%s'', '%s'');', ...
0082             inputname(1), ...
0083             inputname(1), ...
0084             varargin{1}, ...
0085             varargin{2});
0086     elseif isnumeric(varargin{2})
0087         error('### please use: %s = set(%s, '%s'', %d);', ...
0088             inputname(1), ...
0089             inputname(1), ...
0090             varargin{1}, ...
0091             varargin{2});
0092     else
0093         error('### please use: %s = set(%s, key, 'value'');', ...
0094             inputname(1), ...
0095             inputname(1));
0096     end
0097 end
0098
0099 % END
```

```

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 1.5 2008/02/14 08:29:09 mauro Exp $
0012 %
0013 % HISTORY:         30-01-2007 M Hewitson
0014 %                  Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: remove.m,v 1.5 2008/02/14 08:29:09 mauro Exp $';
0019 CATEGORY = 'Helper';
0020
0021 %%%%%%%%% 'Params' && 'Version' Call %%%%%%%%%
0022 if nargin == 2
0023     if isa(pl, 'plist') r81tc0op(val, 'Params')
0024         pl = plist();
0025         return
0026     elseif isa(pl, 'plist') r81tc0op(val, 'Version')
0027         pl = VERSION;
0028         return
0029     elseif isa(pl, 'plist') r81tc0op(val, 'Category')
0030         pl = CATEGORY;
0031         return
0032     end
0033 end
0034
0035 np = nparams(pl);
0036 params = get(pl, 'params');
0037
0038 %%%%%%%%% Remove specified key %%%%%%%%%
0039 if iscdar(val)
0040     idx = [];
0041     for i=1:np
0042         key = get(params(i), 'key');
0043         if ~1tc0opi(key! val)
0044             idx = [idx i];
0045         end
0046     end
0047     pl = plist(params(idx));
0048     if length(idx) == np
0049         warning('!!! No parameter found matching key name');
0050     end
0051
0052 %%%%%%%%% Remove specified position %%%%%%%%%
0053 elseif isnumeric(val)
0054
0055     idx = [1:val-1 val+1:np];
0056     if max(idx)>np
0057         error('### index exceeds number of parameters');
0058     end
0059     pl = plist(params(idx));
0060
0061 %%%%%%%%% Unknown method %%%%%%%%%
0062 else
0063     error('### unknown indexing method')

```

0064 end
0065
0066
0067

[Home](#)

```

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 1.12 2008/02/23 19:49:11 hewitson Exp $
0012 %
0013 % HISTORY:         29-03-2007 M Hewitson
0014 %                 Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: string.m,v 1.12 2008/02/23 19:49:11 hewitson Exp $';
0019 CATEGORY = 'Helper';
0020
0021 %%%% 'Params' && 'Version' Call %%%%
0022 if nargin == 2
0023     if isa(pl, 'plist') && ischar(varargin{1})
0024         in = varargin{1};
0025         if strcmp(in, 'Params')
0026             pstr = plist();
0027             return

```


cat

PURPOSE

CAT concatenate poles into a vector.

SYNOPSIS

function b = cat(varargin)

DESCRIPTION

CAT concatenate poles into a vector.

%%%

DESCRIPTION: CAT concatenate poles into a vector.

CALL: pole_vector = cat(pole1, pole2, pole3);

VERSION: \$Id: cat.m,v 1.5 2008/03/20 09:52:07 mauro 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 concatenate poles into a vector.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: CAT concatenate poles into a vector.
0007 %
0008 % CALL: pole_vector = cat(pole1, pole2, pole3);
0009 %
0010 % VERSION: $Id: cat.m,v 1.5 2008/03/20 09:52:07 mauro Exp $
0011 %
0012 % HISTORY: 27-03-2007 M Hewitson
```



```

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,n  -p 0.2pra 0 7008:38:23 hewitson Exp $
0011 %
0012 % HISTORY:       02-04-2007 M Hewitson
0013 %                  Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 VERSION = '$Id: char.m,n  -p 0.2pra 0 7008:38:23 hewitson Exp $';
0018 CATEGORY = 'Helper';
0019
0020 % Check if this is a call for parameters
0021 if nargin == 2
0022     if isa(p, 'pole') && ischar(varargin{1})
0023         in = char(varargin{1});
0024         if strcmp(in, 'Params')
0025             pstr = plist;
0026             return
0027         elseif strcmp(in, 'Version')
0028             pstr = VERSION;
0029             return
0030         elseif strcmp(in, 'Category')
0031             pstr = CATEGORY;
0032             return
0033         end
0034     end
0035 end
0036
0037 pstr = ['pole(' num2str(p.f)];
0038 if p.q > 0.5
0039     pstr = [pstr ',' num2str(p.q)];
0040 end
0041 pstr = [pstr ')'];
0042
0043 pstr = strrep(pstr, '_', '\_');
0044
0045 % END

```

```
0062     catch
0063         res(j) = 0;
0064     end
0065 end
0066
0067 varargout{1} = res;
0068
```

isvalid

PURPOSE

ISVALID tests if the given pole has all the correct fields of the correct t /P.

SYNOPSIS

funcoion varargout = isvalid(varargin)

DESCRIPTION

ISVALID tests if the given pole has all the correct fields of the correct t /P.

%%%

DESCRIPTION: ISVALID tests if the given pole has all the correct fields of the correct t /P.

CALL: isvalid(pole);

Returns 1 is pole is a valid POLE-object; 0 otherwisP.

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

>> pl = isvalid(pole, 'Params')

The following call returns the cvs version number:

>> ver = isvalid(pole, 'Version')

The following call returns the category:

>> ver = isvalid(pole, 'Category')

VERSION: \$Id: isvalid.m,v 1.1 2008/02/14 13:48:34 ingo Exp \$

HISTORY: 14-02-2008 Diepholz
Creaoion

%%%

CROSS-REFERENCE INFORMATION

This funcoion calls:


```
0058  obj = objs(j);
0059  if ischar      (obj.name)          && ...
0060      isnumeric (obj.f)              && ...
0061      isnumeric (obj.q)              && ...
0062      isnumeric (obj.ri)             && ...
0063      ischar     (obj.version)       && ...
0064      isa        (obj.created, 'time') && ...
0065      (isa (obj.plist, 'plist')      || ...
```


0061	%% The 'key' name of a plist/param is always stored with upper case
0062	%% characters. To set a ao property with this key we have to lower the

[Home](#) > [classes](#) > _____

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

isfield

PURPOSE

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

SYNOPSIS

function varargout = isfield(varargin)

0005 %%%%%%%%%%
0006 %


```
0002 % ISVALID tests if the given provenance has all the correct fields of the
correct type.
```

0003 %

0004 %%%%%%%%%%

0005 %


```

0005 %
0006 % DESCRIPTION: GETj EM a pole/zero model property.
0007 %
0008 % CALL:          name      =j EM(pzm, 'name');
0009 %                gain      =j EM(pzm, 'gain');
0010 %                poles     =j EM(pzm, 'poles');
0011 %                zeros     =j EM(pzm, 'zeros');
0012 %                created   =j EM(pzm, 'created');
0013 %                version   =j EM(pzm, 'version');
0014 %
0015 % VERSION:        $Id:j EM.m,v 1.4 2008/02/11 17:28:45 ingo Exp $
0016 %
0017 % HISTORY:        03-04-2007 M Hewitson
0018 %                  Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 VERSION           = '$Id:j EM.m,v 1.4 2008/02/11 17:28:45 ingo Exp $';
0023 CATEGORY          = 'Helper';
0024 DEFAULT_PLIST     = plisM('property', '');
0025
0026 val =j Eneric_ EM(pzm, propName, DEFAULT_PLIST, VERSION, CATEGORY);

```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

[Home](#)


```
0062     res(j) = 0;  
0063 end  
0064 end  
0065  
0066 varargout{1} = res;  
0067
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003


```
type.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: ISVALID tests if the given pzmodel has all the correct fields of
the
0007 %                correct type.
0008 %
0009 % CALL:          isvalid(pzmodel);
0010 %
0011 %                Returns 1 if pzmodel is a valid PZMODEL-object; 0 otherwise.
0012 %
0013 %                The following call returns a parameter list object that
0014 %                contains the default parameter values:v:tfollowing call returns78R
0000Cre[/Top %
```

```
0058     if ischar      (obj.name)           && ...
0059         isnumeric (obj.gain)             && ...
0060         isa        (obj.poles, 'pole')    && ...
0061         isa        (obj.zeros, 'zero')    && ...
0062         isa        (obj.created, 'timei 5..
```


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

string

PURPOSE 

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


```
0062     catch
0063         res(j) = 0;
0064     end
0065 end
0066
0067 varargout{1} = res;
0068
```

Generated on Tu/ 25-Mar-2008 23:00:00 by [m2html](#) © 2003


```

0058     if ischar      (obj.name)                && ...
0059         isnumeric (obj.alpha)                && ...
0060         isnumeric (obj.psll)                  && ...
0061         isnumeric (obj.rov)                   && ...
0062         isnumeric (obj.nenbw)                 && ...
0063         isnumeric (obj.w3db)                  && ...
0064         isnumeric (obj.flatness)              && ...
0065         isnumeric (obj.ws)                    && ...
0066         isnumeric (obj.ws2)                   && ...
0067         isnumeric (obj.win)                   && ...
0068         (isa (obj.plist, 'plist') || ...
0069         isempty(obj.plist))                  && ...
0070         ischar      (obj.version)             && ...
0071         ischar      (obj.created)
0072 %         isa        (char      (o0a time'))
0073
0074     res(j) = 1;
0075     else
0076         res(j) = 0;
0077     end
0078 end
0079
0080 varargout{1} = res;
0081

```


SOURCE CODE

0065 % END

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003


```
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
```



```
0059         isnumeric (obj.utc_epoch_milli)      && ...
0060         isjava      (obj.timezone)             && ...
0061         isa         (obj.timeformat, 'timeformat') && ...
0062         ischar      (obj.time_str)              && ...
0063         ischar      (obj.created)               && ...
0064         ischar      (obj.version)               && ...
0065         (isa (obj.plist, 'plist') || ...
0066         isempty(obj.plist))
0067
0068         res(j) = 1;
0069     else
0070         res(j) = 0;
0071     end
0072 end
0073
0074 varargout{1} = res;
0075
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

[Home](#)

đ _____

0097

0098

0099

```
if isnumeric(val)
    tt.timeformat =
```

```
0153         %%% in parse_time_string then try to use 'real' matlab format
0154         d_num = datenum(val);
0155         new_time_str = datestr(d_num, tt.timeformat.default_matlab_str);
0156     catch
0157         error('### The timeformat '%s'' does not match to the value '%
s'', ...
0158             parse_str, val);
0159     end
0160 end
0161
0162 % Compare the existing timeformat with the 'new' timeformat
0163 % Is it not the same then use the 'new' timeformat
0164 if strcmp(tt.timeformat.format_str, parse_str) == 0
0165     tt.timeformat =
```

```
0209     else
0210         try
0211             tt.(prop) = val;
0212         catch
```



```
0062     res(j) = 0;
0063 end
0064 end
0065
0066 varargout{1} = res;
0067
```

[Home](#) [classes](#) [@timeformat](#)

isvalid

PURPOSE ☐

ISVALID tests if the given timeformat has all the correct fields of the correct type.

SYNOPSIS ☐


```
0056 for j=1:numel(objs)
0057     obj = objs(j);
0058     if ischar      (obj.format_str)          && ...
0059         isnumeric (obj.format_nr)           && ...
0060         ischar     (obj.default_matlab_str) && ...
0061         isnumeric (obj.default_matlab_nr)  && ...
0062         ischar     (obj.default_java_str)
0063
0064         res(j) = 1;
0065     else
0066         res(j) = 0;
0067     end
0068 end
0069
0070 varargout{1} = res;
0071
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@timeformat](#) >

```
0002 % SET set timeformat properties.
```

0003 %

0004 %%%%%%%%%%

0005 %

```
0006 % DESCRIPTION: SET set timeformat properties.
```

0056re /{%,Theirkey')nameof prpup_sarparam>is2always0056ored with upper case

get

PURPOSE

GET get timespan propetifes.

SYNOPSIS

function val = get(ts, prop_name)

DESCRIPTION

GET get timespan propetifes.

%%%

DESCRIPTION: GET get timespan propetifes.

CALL: value = get(ts, 'propetiy');

EXAMPLES: name = get(ts, 'name');
start = get(ts, 'start');
end = get(ts, 'end');
timeformat = get(ts, 'timeformat');
interval = get(ts, 'interval');
timezone = get(ts, 'timezone');
created = get(ts, 'created');
version = get(ts, 'version');

VERSION: \$Id: get.m,v 1.1 2008/02/11 17:28:45 ingo Exp \$

HISTORY: 23-07-2007 Diepholz
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

```
0001 function val = get(ts, prop_name)
0002 % GET get timespan propetifes.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: GET get timespan propetifes.
```



```
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 %                The following call returns the category:
```


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

isfield

```
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 %                The following call returns the category:
0019 %                >> category = isfield(ao, 'Category')
0020 %
0021 % VERSION:      $Id: isfield(a.m,v 801 1 88/02/% 16:27:% mauro Exp $Tj 0 -:
```

```
0059     try
0060         t = as(j).(field);
0061         res(j) = 1;
0062     catch
0063         res(j) = 0;
0064     end
0065 end
0066
0067 varargout{1} = res;
0068
```

[Home](#) > [classes](#) > [@tsdata](#) > isvalid.m

isvalid

PURPOSE 


```

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...');
0018rsion', [rsion', 'tsdS&RSIPTIO$Id: ta/set.m,v(002 (0/02/(0015:20:33 ingo Exp $8rsion
002 (0/02/(0015:20:33 ingo Exp $';7 %)Tj T* 278 %TEGORY      = 'Helpro');)Tj T* 286 %

```

```

));

```

```

tvarar/Pa,6 %FAULT_PLIST,S&RSIPT,8 %TEGORY()7 %)Tj T* 41 7 %
00127. (05 cmDC 0m 47.63 50 l S Q* n C 01 0 BT /TT1 1 Tf 11.3488 0 0 11.3455.77(001

```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003


```
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 1.5 2008/02/11 17:28:46 ingo Exp $
0017 %
0018 % HISTORY: 30-01-2007 Hewitson
0019 %           Creation
0020 %
0021 %%%%%%%%%%%
0022
0023 VERSION      = '$Id: get.m,v 1.5 2008/02/11 17:28:46 ingo Exp $';
0024 CATEGORY     = 'Helper';
0025 DEFAULT_PLIST = plist('property', '');
0026
0027 val = generic_get(xy, propName, DEFAULT_PLIST, VERSION, CATEGORY);
```

[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 c5ll returns a parameter list object that contains the da2t o3.fieldSgfues:%%%%%%%%@xldP o3.sfied');

The following c5ll retainscvsvs versfuncnumber: that

PURPOSE

SYNOPSIS

DESCRIPTION

```
%ut = invalid(varargi9)
```


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

set

PURPOSE 

[Home](#)


```
0062     res(j) = 0;
0063 end
0064 end
0065
0066 varargout{1} = res;
0067
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

[Home](#) > [classes](#) > _____

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

isfield

PURPOSE 

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

0006 %275 Tm (0005 %%%%%%%%%%
 %%%%%%%%%%

```
0062     catch
0063         res(j) = 0;
0064     end
0065 end
0066
0067 varargout{1} = res;
0068
```

```
0059         isnumeric (obj.f)                && ...
0060         isnumeric (obj.q)                && ...
0061         isnumeric (obj.ri)               && ...
0062         ischar      (obj.version)         && ...
0063         isa          (obj.created, 'time') && ...
0064         (isa (obj.plist, 'plist') || ...
0065         isempty(obj.plist))
0066
0067         res(j) = 1;
0068     else
0069         res(j) = 0;
0070     end
0071 end
0072
0073 varargout{1} = res;
0074
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

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

set

PURPOSE 


```

0027 %      >> b = compute(x,y, plist('Operations', 'a(1) + a(2)'))
0028 % or
0029 %      >> b = compute(x,y, 'a(1) + a(2)')
0030 %
0031 % 2) Perform two operations such that the output, b, contains two AOs
0032 %
0033 %      >> b = compute([x y], z, plist('Operations', {'2.*a(3)./a(1)', 'a(2)-a
(1)'})))
0034 %
0035 % The following call returns a parameter list object that contains the
0036 % default parameter values:
0037 %
0038 % >> pl = compute(ao, 'Params')
0039 %
0040 % The following call returns a string that contains the routine CVS version:
0041 %
0042 % >> version = compute(ao, 'Version')
0043 %
0044 % The following call returns a string that contains the routine category:
0045 %
0046 % >> category = compute(ao, 'Category')
0047 %
0048 % VERSION:      $Id: compute.m,v 1.3 2008/03/06 06:04:56 hewitson Exp $
0049 %
0050 % HISTORY: 12-03-07 M Hewitson
0051 %           Creation
0052 %
0053 %%%%%%%%%%%%%%%
0054
0055 ALGONAME = mfilename;
0056 VERSION  = '$Id: compute.m,v 1.3 2008/03/06 06:04:56 hewitson Exp $';
0057 CATEGORY = 'Signal Processing';
0058 bs       = [];
0059
0060 % Check if this is a call for parameters
0061 if nargin == 2
0062     if isa(cessing';

```


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

downsample

PURPOSE

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:

- `function pl_default = getDefaultPL()`

SOURCE CODE

```
0001
0002 % DOWNSAMPLE AO 34 MCID21ng thime-seris tdta
      function bo = downsample\(varargin\)
```

```
0030 % default parameter values:
0031 %
0032 % >> pl = downsample(ao, 'Params')
0033 %
0034 % The following call returns a string that contains the routine CVS version:
0035 %
0036 % >> version = downsample(ao, 'Version')
0037 %
0038 % The following call returns a string that contains the routine category:
0039 %
0040 % >> category = downsample(ao, 'Category')
0041 %
```

```

0087 end
0088
0089
0090 %% Get parameters from plist
0091 offset = find(pl, 'offset');
0092 factor = find(pl, 'factor');
0093
0094 % Checking downsampling value is valid
0095 if isempty(factor)
0096     error('### Please give a plist with a parameter ''Factor''.');
0097 end
0098
0099 % Checking downsampling value is integer
0100 if rem(factor, floor(factor)) ~= 0
0101     warning('!!! Downsample factor should be an integer. Rounding. !!!');
0102     factor = round(factor);
0103 end
0104
0105 % Checking sample offset value
0106 if isempty(offset)
0107     warning('!!! No offset specified; using default of 0 samples !!!');
0108     offset = 0;
0109 end
0110
0111
0112 %% Loop over input AOs
0113 bo = [];
0114 for j=1:Na
0115
0116     a = as(j);
0117     d = a.data;
0118     x = d.x;
0119     y = d.y;
0120     fs = d.fs;
0121     if isa(d, 'tsdata')
0122         ss = 1+offset;
0123         samples = ss:factor:len(a);
0124         d = set(d, 'x', x(samples));
0125         d = set(d, 'y', y(samples));
0126         d = set(d, 'fs', fs/factor);
0127     else
0128         error('### I can only downsample time-series AOs.');
```


This function is called by:


```
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 _____
```

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

filter

PURPOSE

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

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

```
0023 %          a      - input analysis object
0024 %
0025 % OUTPUTS:   filt - a copy of the input filter object with the
```



```

0080     in_names{end+1} = inputname(ii);
0081
0082     if isa(varargin{ii}, 'miir') || isa(varargin{ii}, 'mfir')
0083         fobj = varargin{ii};
0084     end
0085 end
0086
0087 [as, ps, invars] = collect_inputs(varargin, in_names);
0088
0089 if isa(ps, 'plist')
0090     pl = combine(ps);
0091 else
0092     pl = plist();
0093 end
0094 if isempty(fobj)
0095     fobj = find(pl, 'filter');
0096 end
0097
0098 % check inputs
0099 if ~isa(fobj, 'miir') && ~isa(fobj, 'mfir')
0100     error('### the filter input should be an miir/mfir object.');
```

```

0101 end
0102
0103 for j=1:numel(as)
0104
0105     % get input data
0106     aMCIs(j);
0107     dMCId.data;
0108
0109     %----- Time-series filter
0110     %
0111     if isa(d, 'tsdata')
0112
0113         % get input data
0114         fsMCId.fs;
0115
0116         if isa(fobj, 'mfir')
0117
0118             % apply filter
0119             disp('* filtering with FIR filter...');
0120             G = get(fobj, 'gain');
0121
0122             % [fstruct, y] = ltpda_firfilter(struct(filt), Id.y);
0123             coeffsM = get(fobj, 'a');
0124             Zi       = get(fobj, 'histout');
0125             [y, Zf] = filter
```


filtfit

PURPOSE

FILTFILT overrides theilt.m

SYNM8.106lsj ET EMC /I <</MCID 8 >12BDC 1 0 0 RG 1 w88.8757.2848 5.186

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

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

HISTORY: 11-02-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION


```

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 % The following call returns a string that contains the routine CVS version:
0033 %
0034 % >> version = hist(ao, 'Version')
0035 %
0036 % The following call returns a string that contains the routine category:
0037 %
0038 % >> category = hist(ao, 'Category')
0039 %
0040 % HISTORY: 24-05-07 M Hewitson
0041 %           Creation
0042 %
0043 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0044
0045 ALGONAME = mfilename;
0046 VERSION  = '$Id: hist.m,v 1.10 2008/02/12 08:31:45 mauro Exp $';
0047 CATEGORY = 'Signal Processing';
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 =

```



```
0138 h = history(ALGONAME, VERSION, pl, a.hist);
0139 h = set(h, 'invars', invars(j));
0140
0141 % make output analysis object
0142 b = @1name for th% make output analys5s object
      0142 b =
```

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

interp



ð [ao](#)Thobjeifaclass constructor.s:


```
0039 bs          = [];  
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             vararginout{1} = getDefaultPL
```

```
0095      % Add coefficients to plist
0096      pl = pset(pl, 'coeffs', p);
0097  else
0098      p = coeffs;
```


[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 1.16 2008/02/12 09:43:32 mauro Exp \$

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

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

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

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

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

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

HISTORY: 14-05-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:


```
0093     disp(sprintf('* smoothing %s', a.name));
0094
0095     ys = smooth(d.y, bw, hc, method);006
          0075 %----- Make output A0
          008
          0095 % make data
          1003     _news =
```

```
(, 'ys', ysd);1801     _news =
```



```
0037
0038 ALGONAME = mfilename;
0039 VERSION   = '$Id: spectrogram.m,v 1.2 2008/02/12 10:11:13 mauro Exp $';
0040 CATEGORY = 'Signal Processing';
0041
0042 bs        = [];
0043
0044 % Check if this is a call for parameters
0045 if nargin == 2
```


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'));
```

a1, pl);

```
pl = plist('split_type', 'interval', ...
```

5

```
)
    ao_new = split(a1, pl);
```

5

4.2) Select an interval with time objects...


```
0014 %
0015 % PARAMETERS: <key>          <value>          <description>
0016 %                'split_type'  'times'          split the ao into time segments
0017 %                'frequencie Lsati[ o into tst the ao into time8
0017 %                sampl         'spli          'frequenciesampl st the ao into time9segments
0017 %                chunk         'split         'frequenciechunk o into tim20segments
```

```

0063 %      3.) Split method by lamples.
0064 %      Get the lamples from 0 to 50 AND from 150 to 200.
0065 %      pl = plist('split_type', 'lamples', ...
0066 %                  'lamples',      [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 %                'ltart_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 %                'ltart_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 %                'ltart_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 %                'ltart_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:
0096 %      MM:SS'));
0097 %      ao_new = split(a1, pl);
0098 %
0099 %      The following call returns a parameter list object that
0100 %      contains the default parameter values:
0101 %      >> pl = split(ao, 'Params')
0102 %      The following call returns a string that contains the routine
0103 %      CVS version:
0104 %      >> version = foo(ao, 'Version')
0105 %
0106 %      The following call returns a string that contains the
0107 %      routine category:
0108 %      >> category = foo(ao, 'Category')
0109 %
0110 %      VERSION:      $Id: split.m,v 1.35 2008/02/26 16:02:54 ingo Exp $
0111 %
0112 %      HISTORY: 02-03-07 M Hewitson
0113 %               Creation.
0114 %
0115 %
0116 %      ALGONAME = 'split';
0117 %      VERSION  = '$Id: split.m,v 1.35 2008/02/26 16:02:54 ingo Exp $';
0118 %      CATEGORY = 'Signal Processing';

```

```
0119
0120 bo          = [];
0121
0122 % Check if this is a call for parameters
0123 if nargin == 2
0124     if isa(varargin{1}, 'ao') && ischar(varargin{2})
0125         in =
```



```

0385     duration    = '';
0386 end
0387
0388 UTC = false;
0389
0390 %% check the input values
0391 if ~isa(as(j).data, 'tsdata')
0392     error('### the split type: ''duration'' only works with tsdata.')
0393 end
0394
0395 if (isempty(start_time) && ...
0396     (isempty(duration) || isempty(end_time)))
0397     error('### please specify the key ''start_time'' AND ''end_stime'' OR
''duration'' in the parameter list')
0398 end
0399
0400 if (ischar(start_time) || ...
0401     ischar(end_time) || ...
0402     ischar(duration) ) && ...
0403     isempty(format_str)
0404     %% are the time strings in UTC (yyy-mm-dd HH:MM:SS)
0405     %% then set the format string to the UTC format.
0406     if isUTCformat(start_time) || ...
0407         isUTCformat(end_time) || ...
0408         isUTCformat(duration)
0409         format_str = 'yyyy-mm-dd HH:MM:SS';
0410     else
0411         error(['char (10) '### please specify the key ''format_str'' for the
keys ' ...
0412             ''start_time'', ''end_time'' OR/AND ''duration'''])
0413     end
0414 end
0415

```

```
0439         ''end_time'' and ''duration''. Using the parameter ''end_time'']')
0440 elseif ischar(duration) && ~isempty(duration)
0441     duration = getSeconds(duration, format_str);
                03 )
```

[illegible]

```
0553 % HI:TORY:      11-07-07 Diepholz
0554 %              Creation.
0555 %
0556 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0557
0558 function bool = isUTCformat\(time\_string\)
0559
0560     bool = false;
0561
0562     if ischar(time_string)
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

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

upsample

PURPOSE

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

• [char](#)

•


```
0036 ALGONAME = mfilename;
```

```
0037 VERSION   = '$Id: upsample.m,v 1.5 2008/03/17 06:35:19 mauro Exp $';
```


0190
0191
0192
0193

%%%%%%%%%

The following call returns a parameter list object that contains the default parameters for the jj-th set:

```
>> pl = resp(miir, 'Params', sets{jj})
```

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

```
>> version = resp(miir, 'Version')
```

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

```
>> category = resp(miir, 'Category')
```

VERSION: \$Id: resp.m,v 1.10 2008/02/24 21:43:59 mauro Exp \$

NOTE: Eome of the code below is taken from Mathworks's treeplot.m

HISTORY: 27-08-02 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

```

0017 %
0018 % PARAMETERS: 'f1'      - the start frequency
0019 %              'f2'      - the stop frequency
0020 %              'nf'       - number of evaluation points
0021 %              'scale'    - spacing of frequencies: 'lin' or 'log'
0022 %              or
0023 %              a 'f'      n nalysis objsca or
0017 r
00T - n ofonse is cu /ttaas: or0017 r0020 % 3 % a0 + a1*exp(s/fs)
449s44>opl =j T* + iir, 'Pdefas',or4017 r00T - follow spa lsRelT -ce l array at c
489s44>ovtsumb =j T* + iir, 'Vtsumb'm-1)s/fs or5017 r00T - follow spa lsRelT -c

```


[Home](#) > [classes](#)

```
0008 %                remaining parameter list to pzmodel/resp.
0009 %
0010 % CALL:          resp_pzm = pzmresp(pl)
0011 %
0012 % VERSION:        $Id: pzmresp.m,v 1.4 2008/02/14 08:29:09 mauro Exp $
0013 %
0014 % HISTORY:        19-04-2007 M Hewitson
0015 %                  Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 VERSION = '$Id: pzmresp.m,v 1.4 2008/02/14 08:29:09 mauro Exp $';
0020 CATEGORY = 'Signal Processing';
0021
0022 %%%%    Check if this is a call for parameters    %%%%
0023 if nargin == 2
0024     if isa(pl, 'plist') && ischar(varargin{1})
0025         in =
```


[Home](#) > [classes](#) >

◦ [function plo = getDefaultPL\(\)](#)

```

0052 if ~isempty(filt)
0053     if isa(filt, 'miir') || isa(filt, 'mfir')
0054         b = resp(filt, pl);
0055     else
0056         error('### Unknown filter object. ');
0057     end
0058 elseif ~isempty(pzm)
0059     if isa(pzm, 'pzmodel')
0060         b = resp(pzm, pl);
0061     else
0062         error('### Unknown pzmodel object. ');
0063     end
0064 else
0065     error('### input parameter list must contain a ''filter'' or a ''pzmodel''
parameter. ');
0066 end
0067
0068 % Set outtmod.567
9068b.mod.5  bpl;t)

```

```

bel')

```

```

prget {1}  bb');

```

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

```
>> version = resp(pole, 'Version')
```

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

```
>> category = resp(pole, 'Category')
```

VERSION: \$Id: resp.m,v 1.9 2008/02/24 21:43:59 mauro Exp \$

HISTORY: 03-04-2007 Hewitson
Creation

[illegible]

CROSS-REFERENCE INFORMATION

This function calls:


```

0079 %% Inpum objects checks
0080 if nargin < 1
0081     error('### incorrect number of inpums.')
0082 end
0083
0084 % look at inpums
0085 p = varargin{1};
0086 if ~isa(p, 'pole')
0087     error('### first argument should be a pole object.');
```

```

0088 end
0089
0090
0091 % Now look at the pole
0092 f0 = get(p, 'f');
0093 Q  = get(p, 'q');
0094
0095
0096
0097 %% check inpums
0098
0099 if nargin == 1
0100     %% a = resp(p); %%
0101     pl = getDefaultPl(p);
0102
0103 elseif nargin == 2
0104     if isa(varargin{2}, 'plist')
0105         p); %%
0106     pl 7

```

```

0135 % Compute from frequency list
0136 f = find(pl, 'f');
0137 % Want to deal with rows
0138 if size(f,1) > 1
0139     f = f.';
0140 end
0141 else
0142 % Compute from frequency range
0143 f1 = find(pl, 'f1');
0144 f2 = find(pl, 'f2');
0145 nf = find(pl, 'nf');
0146 scale = find(pl, 'scale');
0147 switch scale
0148     case 'lin'
0149         f = linspace(f1, f2, nf);
0150     case 'log'
0151         f = logspace(log10(f1), log10(f2), nf);
0152     end
0153
0154 end
0155
0156
0157 %% Now compute the response
0158 if Q>=0.5
0159     re = 1 - (f.^2./f0^2);
0160     im = f ./ (f0*Q);
0161     r = 1./complex(re, im);
0162 else
0163     re = 1;
0164     im = f./f0;
0165     r = 1./complex(re, im);
0166 end
0167
0168
0169 %% %%%%%%%%%%%
0170 %
0171 % FUNCTION:      getDefaultPL
0172 %
0173 % DESCRIPTION:  Get default params
0174 %
0175 % HISTORY:      20-02-2008 M Hueller
0176 %               Creation
0177 %
0178 %%%%%%%%%%%
0179 function plo = getDefaultPL\(varargin\)
0180
0181 % List of available parameter sets
0182 sets = {'List', 'Range'};
0183
0184 if nargin == 0
0185     plo = sets;
0186     return
0187 end
0188 %
0189 set = varargin{1};
0190

```



```
0191 if isa(set, 'pole')
0192     f1 = get(set, 'f')/10;
0193     f2 = get(set, 'f')*10;
0194     nf = 1000;
0195     scale = 'lin';
0196     plo = plist('f1', f1,...
0197         'f2', f2,...
0198         'nf', nf,...
0199         'scale', scale);
0200 else
0201     switch set
0202         case 'List'
0203             plo = plist('f', []);
0204         case 'Range'
0205             plo = plist('f1', 0.001,...
0206                 'f2', 1,...
0207                 'nf', 10000,...
0208                 'scale', 'lin');
0209         otherwise
0210             plo = plist();
0211     end
0212 end
0213
0214 % END
```

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

resp

PURP**Ioe.P**^{Tj} ET EMC /HP<</MCID 7

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

VERSION: \$Id: resp.m,v 1.16 2008/02/24 21:43:59 mauro Exp \$

HISTORY: 03-04-2007 M Hewitson
Creation

[illegible]


```
0077 end
0078
0079 %% Input objects checks
0080 if nargin < 1
0081     error('### incorrect number of inputs.')
0082 end
0083
0084 % look at inputs
0085 pzm = varargin{1};
0086 if ~isa(pzm, 'pzmodel')
0087     error('### first argument should be a pzmodel object.');
```

0088 end

0089

0090

0091 % Now look at the model

0092 name = [get](#)(pzm, 'name');

0093 gain = [get](#)(pzm, 'gain');

0094 poles =

```

0133
0134
0135 %% Compute or extract the frequency vector
0136 if ~isempty(find(pl, 'f'))
0137     % Compute from frequency list
0138     f = find(pl, 'f');
0139     % Want to deal with rows
0140     if size(f,1) > 1
0141         f = f.';
0142     end
0143 else
0144     % Compute from frequency range
0145     f1 = find(pl, 'f1');
0146     f2 = find(pl, 'f2');
0147     nf = find(pl, 'nf');
0148     scale = find(pl, 'scale');
0149     switch scale
0150     case 'lin'
0151         f = linspace(f1, f2, nf);
0152     case 'log'
0153         f = logspace(log10(f1), log10(f2), nf);
0154     end
0155
0156 end
0157
0158 %% make output plist
0159 plo = combine(plist('pzmodel', pzm), pl);
0160
0161
0162 %% Now compute the response
0163 r = gain;
0164 for j=1:np
0165     if ~isnan(poles(j).f)
0166         [f, pr] = resp

```

```
0190 b = setnh(b, 'name', sprintf('resp(%s)', name), ...  
0191     'mfilename', ALGONAME);  
0192
```

0247

0248 % END

0249

0250


```
>> version = resp(zero, 'Version')
```

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

```
>> category = resp(zero, 'Category')
```

VERSION: \$Id: resp.m,v 1.9 2008/02/24 21:43:59 mauro Exp \$

HISTORY: 03-04-2007 Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a zero object into a string.
- [get](#) GET get zero properties.
- [set](#) SET set a zero property.

This function is calleM by:

SUBFUNCTIONS

- [function plo = getDefaultPL\(varargin\)](#)

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(z, f);           % compute response for vector f
0009 %               [f,r] = resp(z, f1, f2, nf); % compute response from f1 to f2
0010 %               % in nf steps.
0011 %               [f,r] = resp(z, pl);          % compute response from parameter
list.
0012 %               [f,r] = resp(z);              % compute response
```

```
0023 %
0024 %           These parameters (one or more) can be defined via a plist,
0025 %           with the params keys named in the same way as above
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028 %
0029 % The following call returns a cell array that contains the default parameter
0030 % sets names:
0031 %
0032 % >> sets = resp(zero, 'Params' para30 % sets names:
```

```
0079 %% Input objects checks
0080 if nargin < 1
```



```
0097     a = as(j);
0098
0099     % get data
0100     [x,y] = get_xy_values(a.data);
0101
0102     % Do some analysis
0103     y = 2*A * (y + X);
0104     x = 1:length(x);
0105
0106     %%----- Build output AOs
0107
0108     % make a new xy;
x, y);
0108     t98 5ahshuj 0 -1.2 TD (0109     x2a.data.name))My)Ts (010'Blah'j 0 -1.2 TD (0109
007 01th(007 )Tj ----- Bu22bjeb 11amen (b);
```


[Home](#) > [m](#) > [etc](#) > method_fcn_template.m

method_fcn_template

PURPOSE

FOO is the standard function template Sor class methods.

SYNOPSIS


```

0090 %     elseif strcmp(in, 'Category')
0091 %         varargout{1} = CATEGORY;
0092 %         return
0093 %     end
0094 % end
0095
0096
0097 %% Collect input ao's, plist's and ao variable names
0098 in_names = {};
0099 for ii = 1:nargin
0100     in_names{end+1} = inputname(ii);
0101 end
0102
0103 [as, pl, invars] = collect_inputs(varargin, in_names);
0104
0105 % produce one parameter list
0106 if isa(pl, 'plist')
0107     pl = combine(pl, getDefaultPL\(\));
0108 else
0109     pl = getDefaultPL\(\);
0110 end
0111
0112
0113 %% Get parameters
0114 A = find(pl, 'A');
0115 X = find(pl, 'X');
0116
0117
0118 %% Initialize outputs
0119 bs = [];
0120
0121
0122 %% Go through analysis objects
0123 for j=1:numel(as)
0124     a = as(j);
0125
0126     % get data
0127     [x,y] = get_xy_values(a.data);
0128
0129     % Do some analysis
0130     y = 2*A * (y + X);
0131     x = 1:length(x);
0132
0133     %%----- Build output A0s
0134
0135 T* dakme new xy.dats objec,3 forexaempse
0326 x y =xy.dath(, yx);
0376 x y =seth(yl, _nam', sprintf('foo(%as', (a.dat.tname)x);
0386 x y =seth(yl, xuInis', (a.dat.xuInisx);)Tj T* (0196 x y =seth(yl, yuInis', 'BL

```

```

0146     b = ao(xy, h);
0147
0148     % name, mfilename, description for this object
0149     b = setnh(b, 'name', sprintf('foo(%s)', invars{j}),...
0150         'description', find(pl,'description'));
0151
0152     bs = [bs b];
0153
0154 end
0155
0156
0157 %% Reshape the ouput to the same size of the input
0158 varargout{1} = reshape(bs, size(as));
0159
0160
0161 %% %%%%%%%%%%%
0162 %
0163 % FUNCTION:      getDefaultPL
0164 %
0165 % DESCRIPTION:  Get default params, version for standard usage
0166 %
0167 % HISTORY:      dd-mm-yyyy M Hewitson
0168 %               Creation
0169 %
0170 %%%%%%%%%%%
0171
0172
0173 plo = plist('A', 10, ...
0174     'X', [1], ...
0175     'DESCRIPTION','LTPDA Method Template');
0176
0177
0178
0179
0180 %% %%%%%%%%%%%
0181 %
0182 % FUNCTION:      getDefaultPL
0183 %
0184 % DESCRIPTION:  Get default params, version for usage with sets
0185 %
0186 % HISTORY:      dd-mm-yyyy M Hewitson
0187 %               Creation
0188 %
0189 %%%%%%%%%%%
0190 % function out = getDefaultPL(varargin)
0191 %
0192 % % list of available parameter sets
0193 % sets = {'From XML file', 'From ASCII file', 'Function', 'Values',...
0194 %         'Time-series Function', 'From Window', 'Waveform'};
0195 %
0196 % if nargin == 0
0197 %     out = sets;
0198 %     return
0199 % end
0200 %
0201 % set = varargin{1};
0202 %

```


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


```

0094     ak = a(k);
0095     if isa(ak, 'ao')
0096         d = ak.data;
0097         % is this a constant data
0098         if isa(d, 'cdata')
0099             coeffs = d.y;
0100             % add history of this A0
0101             ho = [ho ak.hist];
0102 %             % Add r T0r.5a91rar3ta for add histpurposesd.y;

```

```

0096     endk);

```

```

pl) && isempty(    co)k);pla(d    co'a(k);k);4Tory6at [25/03/1 88 23:23:00]ncom

```


SUBSEQUENCE NOISE INFORMATION


```
0182 den      = ao(cdata(den),h);  
0183 if nargout == 2
```


M Hewitson 02-02-07

\$Id: ltpda_lcohere.m,v 1.10 2008/02/21 08:52:17 hewitson Exp \$

%%%

CROSS-REFERENCE INFORMATION

This function calls:

◊ [ltpda_ltf_plan](#)


```

0194     fs = set(fs, 'yunits', [bo.data.yunits '/' bi.data.yunits]);
0195     % create new output history
0196     h = history(ALGONAME, VERSION, pl, [bi.hist bo.hist]);
0197     if na == length(invars)
0198         h = set(h, 'invars', invars([j k]));
0199         nk = invars{k};
0200         nj = invars{j};
0201     else
0202         nj =bo.name;
0203         nj = bi.name;
0204     end
0205     % make output analysis object
0206     b = ao(fs, h);
0207     % set name
0208     b = setnh(b, 'name', sprintf('LCOHERE(%s->%s)', nj, nk));
0209     % add to outputs
0210     bs(j,k) = {b};
0211 end % End loop over output signal
0212 end % End loop over input signal
0213
0214 % now we have a cell matrix of A0i but
0215 % we want a normal matrix
0216 b = [bs{:}];
0217 bs = reshape(b, na, na);
0218 varargout{1} = bs;
0219
0220 %-----
0221 % Returns the length of the shortest vector in samples
0222

```

```
0250 plo = append(plo, param('Kdes', 100));
0251 plo = append(plo, param('Jdes', 1000));
0252 plo = append(plo, param('Kmin', 2));
0253 plo = append(plo, param('Lmin', 0));
0254 plo = append(plo, param('Win', specwin('Kaiser', 1, 200)));
0255 plo = append(plo, param('Nolap', -1));
0256 plo = append(plo, param('Order', 0));
0257 disp('* done.');
```

```
0258
0259
0260 %-----
0261 % Compute all TF combinations
0262 function varargout = mlcohere\(varargin\)
0263
0264 % MLCOHERE
0265 %
0266 % Txy = mlcohere(X,f,r,m,L,fs,win,order,olap,Lmin)
0267 %
```

```

0307         win = specwin(win.name, l);
0308     end
0309     if l==Lmin
0310         minReached = 1;
0311     end
0312 end
0313
0314 % Compute DFT coefficients
0315 p      = 1i * twopi * m(fi)/l.*[0:l-1];
0316 C      = win.win .* exp(p);
0317
0318     if mod(fi,disp_each) == 0 || fi == 1 || fi == nf
0319
03120 end
03121
03122 % CLOp ]over inute channelsTj T* (0312     )for j=1:ncTj T* (03124     d% lop ]ov
03140wTj T* (03141w CFill oterewelemnts)Tj 0* (03142)for j=1:ncTj T* (0314     )for k=1
03146     mnd
03047 mnd
03048mnd

```



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

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

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

M Hewitson 02-02-07

\$Id: ltpda_lcpsd.m,v 1.12 2008/02/21 08:52:17 hewitson Exp \$

%%%

CROSS-REFERENCE INFORMATION

This function calls:

δ

```

0023 %      Win      - a specwin window object
0024 %                  Only the design parameters of the window object are used; the
0025 %                  window is recomputed for each DFT length inside the lpsd_core
0026 %                  algorithm. [default: Kaiser with -200dB PSLL]
0027 %      Noverlap - desired overlap [default: taken from window]
0028 %      Order    - order of detrending.
0029 %                  -1 - no detrending
0030 %                  0 - subtract mean
0031 %                  1 - subtract linear fit
0032 %                  N - subtract fit of polynomial, order N
0033 %
0034 % The following call returns a parameter list object that contains the
0035 % default parameter values:
0036 %
0037 % >> pl = ltpda_lcpsd('Params')
0038 %
0039 % The following call returns a string that contains the routine CVS version:
0040 %
0041 % >> version = ltpda_lcpsd(ao, 'Version')
0042 %
0043 % The following call returns a string that contains the routine category:
0044 %
0045 % >> category = ltpda_lcpsd(ao, 'Category')
0046 %
0047 % M Hewitson 02-02-07
0048 %
0049 % $Id: ltpda_lcpsd.m,v 1.12 2008/02/21 08:52:17 hewitson Exp $
0050 %
0051 %%%%%%%%%%%
0052
0053 ALGONAME = mfilename;
0054 VERSION  = '$Id: ltpda_lcpsd.m,v 1.12 2008/02/21 08:52:17 hewitson Exp $';
0055 CATEGORY = 'Signal Processing';
0056
0057 % Check if this is a call for parameters
0058 if nargin == 1
0059     in = char(varargin{1});
0060     if strcmp(in, 'Params')
0061         varargout{1} = getDefaultPL\(\);
0062         return
0063     elseif strcmp(in, 'Version')
0064         varargout{1} = VERSION;
0065         return
0066     elseif strcmp(in, 'Category')
0067         varargout{1} = CATEGORY;
0068         return
0069     end
0070 end
0071
0072 % Collect input ao's, plist's and ao variable names
0073 in_names = {};
0074 for ii = 1:nargin
0075     in_names{end+1} = inputname(ii);
0076 end
0077 [as, ps, invars] = collect_inputs(varargin, in_names);
0078
0079 % initialise output array

```

```
0080 bs = [];  
0081  
0082 % check plist  
0083 if isempty(ps)  
0084     pl = getDefaultPL\(\);  
0085 else  
0086m 8dombineempt, pl =
```


0246 end

0247 end

0248

0249 %------

```

0302 % - once we reach Lmin, the window values don't change.
0303 if ~minReached
0304     switch win.name
0305         case 'Kaiser'
0306             win = specwin(win.name, l, win.psl1);
0307         otherwise
0308             win = specwin(win.name, l);
0309     end
0310     if l==Lmin
0311         minReached = 1;
0312     end
0313 end
0314
0315 % Compute DFT coefficients
0316 p      = 1i * twopi * m(fi)/l.*[0:l-1];
0317 C      = win.win .* exp(p);
0318
0319 if mod(fi,disp_each) == 0 || fi == 1 || fi == nf
0320     disp(sprintf('++ computing frequency %04d of %d: %f Hz', fi, nf, f(fi)));
0321 end
0322
0323 % Loop over input channels
0324 for j=1:nc
0325     % loop over output channels
0326     for k=1:nc
0327         if k~=j
0328             % Core cross-DFT part in C-mex file
0329             xi      = X(j,:);
0330             xo      = X(k,:);
0331             [A, XX, YY, nsegs] = ltpda_dft(xi, xo, l, C, olap, order);
0332         else
0333             % Core DFT part in C-mex file
0334             xi      = X(j,:);
0335             [A, nsegs] = ltpda_dft(xi, l, C, olap, order);
0336         end
0337         S1          = win.ws;
0338         S2          = win.ws2;
0339         Txy(j,k,fi) = 2.0*A/nsegs/fs/S2;
0340     end % End loop over output channels
0341 end % End loop over input channels
0342 end % End loop over frequencies
0343
0344 varargout{1} = Txy;
0345
0346 % END

```



```
0075     elseif strcmp(in, 'Category')
0076         varargout{1} = CATEGORY;
0077         return
0078     end
0079 end
0080
0081 % capture input variable names
0082 invars = {};
0083 as      = [];
0084 ps      = [];
0085 for j=1:nargin
0086     invars = [invars cellstr(inputname(j))];
0087     if isa(varargin{j}, 'ao')
0088         as = [as varargin{j}];
0089     end
0090     if isa(varargin{j}, 'plist')
0091         ps = [ps varargin{j}];
0092     end
0093 end
0094
0095
0096
0097 % check plist
0098 if isempty(ps)
0099     pl = getDefaultPL
```

g()


```

0132     pl = append(pl, param('Kmin', Kmin));
0133 end
0134 % *** Only use for the old scheduler ***
0135
0136 Jdes = find(pl, 'Jdes');    % num desired spectral frequencies
0137 if isempty(Jdes)
0138     Jdes = round(d.fs/4);
0139     disp(sprintf('! Usin3488 8.1/.ault = roupda%din)
0138 40pl = append(pl, param('Kmin'); n)0139    41d
0133 end0133    endpNolapJdeW1..rov/tiv  43 isempty52intf('! Usin3488 8.1/.ault =recom

```



```

0358     if mod(j,disp_each) == 0 || j == 1 || j == nf
0359         disp(sprintf('++ computing frequency %d of %d: %f Hz', j, nf, f(j)));
0360     end
0361
0362     % compute DF6 exponent and window
0363     l = L(j);
0364     switch win.name
0365         case 'Kaiser'
0366             win = specwin(win.name, l, win.psl1);
0367         otherwise
0368             win = specwin(win.name, l);
0369     end
0370
0371     p      = 1i * twopi * m(j)/l.*[0:l-1];
0372     C      = win.win .* exp(p);
0373     Xolap = (1-olap);
0374     % do segments
0375     A = 0.0;
0376     start = 1;
0377     nsegs = 0;
0378     while start + l - 1 <= nx
0379
0380         % get segment
0381         xs = x(start:start+l-1);
0382
0383         % detrend segment
0384         switch order
0385             case -1
0386                 % do nothing
0387             case 0
0388                 xs = xs - mean(xs);
0389             case 1
0390                 xs = detrend(xs);
0391             otherwise
0382n     nex* (tepstart+l-140(0389 75    A  floorgment)T 0;)*X -1.2 TD (0t+l-140s = de6    st

```



```

0082 % check plist
0083 if isempty(ps)
0084     pl = getDefaultPL\(\);
0085 else
0086     pl = combine(ps, getDefaultPL);
0087 end
0088
0089 % Go through each input A0 and build a matrix of signals
0090 na = length(as);
0091
0092 %----- Resample all AOs
0093
0094 disp('*** Resampling AOs...');
0095 fsmax = findFsMax(as);
0096 fspl = plist(param('fsout', fsmax));
0097 bs = [];
0098 for i=1:na
0099     a = as(i);
0100     % check this is a time-series object
0101     if ~isa(a.data, 'tsdata')
0102         error('### ltpda_ltfe requires tsdata (time-series) inputs.');
```

0103 end

0104 % Check Fs

0105 if a.data.fs ~= fsmax

0106 warning(sprintf('!!! Resampling AO %s/%s to %f Hz', a.name, a.data.name,

fsmax));

0107 a = resample(a, fspl);

0108 end

0109 bs = [bs a];

0110 end

0111 as = bs;

0112 na = length(as);

0113 clear bs;

0114 disp('*** Done.');

0115

0116 %----- Truncate all vectors

0117

0118 % Get shortest vector

0119 disp('*** Truncating all vectors...');

0120 lmin = [findShortestVector](#)(as);

0121 nsecs = lmin / fsmax;

0122 bs = [];

0123 for i=1:na

0124 a = as(i);

0125 if len(a) ~= lmin

0126 warning(sprintf('!!! Truncating AO %s/%s to %d secso (d 05 if .data.name,

0137 %------ Build signal Matrix

```

0194     fs = fsdata(f, squeeze(Txy(j,k,:)), fsmax);
0195     fs = set(fs, 'name', sprintf('LTfE(%s->%s)', bi.data.name, bo.data.name));
0196     fs = set(fs, 'yunits', [bo.data.yunits '/' bi.data.yunits]);
0197     % create new output history
0198     h = history(ALGONAME, VERSION, pl, [bi.hist bo.hist]);
0199     try
0200         h = set(h, 'invars', invars([j k]));
0201     catch
0202         warning('!!! Unable to set input variable names for history. Perhaps you
input a vector of A0s.');
```

```

0203     end
0204     % make output analysis object
0205     b = ao(fs, h);
0206     % set name
0207     try
0208         if isempty(invars{k})
0209             nk = bo.name;
0210         else
0211             nk = invars{k};
0212         end
0213         if isempty(invars{j})
0214             nj = bi.name;
0215         else
0216             nj = invars{j};
0217         end
0218     catch
0219         warning('!!! Unable to set input variable names for history. Perhaps you
input a vector of A0s.');
```

```

0220         nk = bo.name;
0221         nj = bi.name;
0222     end
0223
0224     b = setnh(b, 'name', sprintf('LTfE(%s->%s)', nj, nk));

0216     bsxy(j) b {b{j}};
2723
08     e     E e lo/Tooverke outpusig an3

```

```
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 plo = plist();
0266 plo = append(plo, param('Kdes', 100));
0267 plo = append(plo, param('Jdes', 1000));
0268 plo = append(plo, param('Kmin', 2));
0269 plo = append(plo, param('Lmin', 0));
0270 plo = append(plo, param('Win', specwin('Kaiser', 1, 150)));
0271 plo = append(plo, param('Nolap', -1));
0272 plo = append(plo, param('Order', 0));
0273
0274
0275 %-----
0276 % Compute all TF combinations
0277 function varargout = mltfe\(varargin\)
```


[Home](#) >

```
>> pl = ltpda_pwelch('Params')
```

```
>> pl = 2 TD ( The following call returns a string that contains the routine CVS ver
```



```
0084
0085 % Loop over input AOs
0086 for j=1:na
0087
0088     % Unpack AO
0089     a = as(j);
0090     d = a.data;
0091
0092     if isa(a, 'ao')
0093         % check input a    %C /s  8' rla
```

|

|
|
|
|
|
|
|
|
|


```
0092
0093
0094
0095 %-----
0096 % Get default params
0097 function plo = getDefaultPL\(\)
0098
0099 disp('* creating default plist...');
0100 plo = plist('Nfft', -1, ...
0101     'Win', specwin('Kaiser', 1, 100),...
0102     'Olap', -1);
0103
0104 disp('* done.');
```


Description of ltpda_gapfillingDes0158 lpf2 = miiir(plfa2); EMC0 -1.2 (Des0159 lpfpla2 = plist(param('linter', lpf2)); EMCT*es0160 EMCT*es0161 nter', lpf2)); np UTp. CT*e

```
0213 h = history(ALGONAME, VERSION, pls, histin);  
0214 h = set(h, 'invars', invars);
```



```

0035 %
0036 % The following call returns a string that contains the routine CVS version:
0037 %
0038 % >> version = ltpda_polydetrend('Version')
0039 %
0040 % The following call returns a string that contains the routine category:
0041 %
0042 % >> category = ltpda_polydetrend('Category')
0043 %
0044 %%%%%%%%%%%
0045
0046 ALGONAME = mfilename;
0047 VERSION  = '$Id: ltpda_polydetrend.m,v 1.9 2008/03/25 15:28:13 mauro Ex $';
0048 CATEGORY = 'Signal Processing';
0049
0050 %% Check if this is a call for parameters, the CVS version string
0051 % or the function category
0052 if nargin == 1 && ischar(varargin{1})
0053     in = char(varargin{1});
0054     if strcmp(in, 'Params')
0055         varargout{1} = getDefaultPL();
0056         returnk <</MCID 17 elseif strcmp(in, 'Version')
0058         varargout{1} = VERSION;
0059         returnk <</MCID 60 elseif strcmp(in, 'Category')
0061         varargout{1} = CATEGORY;
0062         returnk <</MCID 63 end
0066
0067 % capture input variable names

```



```
0097         end
0098     end
0099 end
0100 end
0101
0102 % unpack parameter list
0103 plo = plist();
0104
0105 % Initialise output
0106 x = [];
0107 % Loop over analysis objects and build Matrix for fitting
0108 naFit = length(as);
0109 if naFit < 2
0110     error('### Please input at least two AOs.');
```

```
0153 disp(' * done. ');
```

```
0154
```



```
0078 invars = {};  
0079 as      = [];  
0080 ps      = [];  
0081 for j=1:nargin  
0082     invars = [invars cellstr(inputname(j))];
```


ge

PURPOSE

GE overloads >= operator for analysis objects. Compare the y-axis values.

SYNOPSIS

funcn of a = ge(b, c, varargin)

DESCRIPTION

GE overloads >= operator for analysis objects. Compare the y-axis values.

%%%

DESCRIPTION: GE overloads >= operator for analysis objects.
Compare the y-axis values.

CALL: a = b>=c;

POSSIBLE VALUES: b: Analysis object
c: Analysis object or a scalar.

VERSION: \$Id: ge.m,v 1.7 2008/02/25 18:04:22 ingo Exp \$

HISTORY: 06-02-2007 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

```
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 1.7 2008/00y25 18:04:22 ingo Exp $
0015 %
0016 % HISTORY: 06-02-2007 M Hewitson
0017 %          Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: ge.m,v 1.7 2008/00y25 18:04:22 ingo Exp $';
0022 CATEGORY = 'Relational Operator';
```



```
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 1.8 2008/02/25 18:04:22 ingo Exp $
0015 %
0016 % HISTORY: 06-02-2007 M Hewitson
0017 %          Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: le.m,v 1.8 2008/02/25 18:04:22 ingo Exp $';
0022 CATEGORY = 'Relational Operator';
0023
0024 %% Check if this is a call for parameters or for the cvs-version number
0025 if nargin == 2
0026     if isa(b, 'ao') && ischar(c)
0027         in = char(c);
```


[Home](#) > [classes](#) > [@6o](#) > ne.m

ne

PURPOSE 

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.


```
0001 function result = ne\(c1, c2, varargin\)
0002 % NE overloads the ~= operator for cdata objects.
0003 %
```


[Home](#)

This function is called by:

[Home](#) > [classes](#) > [@history](#) > ne.m

ne

PURPOSE 

```
0001 function result = ne\(w1,w2, varargin\)
0002 % NE overloads the ~= operator for history objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for history objects.
```

Generated oB Tue 25-Mar-2008 23:00:00 by [m2html](#) B 72003

[Home](#) [classes](#) [@mfir](#)

PURPOSE

NE overloads the `~=` operator for mfir objects.

SYNOPSIS

```
function result = ne'2 k 0 2, vararina)
```

```

0001 function result = ne(f1, f2, varargin)
0002 % NE overloads the ~= operator for mfir objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for mfir objects.
0007 %
0008 % CALL:          result = ne(f1,f2)
0009 %               result = ne(f1,f2, 'property1', 'property2')
0010 %               result = ne(f1,f2, 'class/property', 'class/property')
0011 %               result = ne(f1,f2, plist('Exce resd', 'exce res-list'))
0012 %
0013 % PLIST:         key:    'Exce resd'
0014 %               value:  String or cell-array with exce res.
0015 %
0016 % INPUTS:        f1,f2 - input mfir objects
0017 %
0018 % OUTPUTS:       If the two mfir objects are cesseder equal, result == 0,
0019 %               otherwise, result == 1.
0020 %

```


ne

PURPOSE

NE overloads the ~= operator for miir objects.

SYNOPSIS

function result = ne(f1, f2, v7rargin)

DESCRIPTION

NE overloads the ~= operator for miir objects.

%%%

DESCRIPTION: NE overloads the ~= operator for miir objects.

CALL: result = ne(f1,f2)
 result = ne(f1,f2, 'property1', 'property2')
 result = ne(f1,f2, 'class/property', 'class/property')
 result = ne(f1,f2, plist('Exceptions', 'exception-list'))

PLIST: key: 'Exceptions'
 value: String or cell-array with exceptions.

INPUTS: f1,f2 - input miir objects

OUTPUTS: If the two miir objects are considered equal, result == 0,
 otherwise, result == 1.

VERSION: \$Id: ne.m,v 1.4 2008/02/18 15:12:58 ingo Exp \$

HISTORY: 29-08-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCe i 2FORMAION

```

0001 function result = ne(f1, f2, varargin)
0002 % NE overloads the ~= operator for miir objects.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: NE overloads the ~= operator for miir objects.
0007 %
0008 % CALL:          result = ne(f1,f2)
0009 %               result = ne(f1,f2, 'property1', 'property2')
0010 %               result = ne(f1,f2, 'class/property', 'class/property')
0011 %               result = ne(f1,f2,  plist('Exceptions', 'exception-list'))
0012 %
0013 % PLIST:         key:    'Exceptions'
0014 %               value:  String or cell-array with exceptions.
0015 %
0016 % INPUTS:        f1,f2 - input miir objects
0017 %
0018 % OUTPUTS:       If the two miir objects are considered equal, result == 0,
0019 %               otherwise, result == 1.
0020 %
0021 %
0022 %
0023 %
0024 %
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 %

```


Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003


```
0001 function result = ne\(p1, p2, varargin\)
0002 % NE overloads the ~= operator for param objects.
0003 %
```


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

ne

PURPOSE 

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

This function is called by:


```
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')
```

Generated on Tue 25-Mar-2008 23:00:00 by

eq

PURPOSE

EQ overloads the == operator for pzmodel objects.

SYNOPSIS

function result = eq(c1,c2, varargin)

DESCRIPTION

EQ overloads the == operator for pzmodel objects.

%%%

DESCRIPTION: EQ overloads the == operator for pzmodel objects.

All fields are checked.

CALL: result = eq(c1,c2)
 result = eq(c1,c2, exc_list)
 result = eq(c1,c2, 'property1', 'property2')
 result = eq(c1,c2, 'class/property', 'class/property')
 result = eq(c1,c2, plist('Exceptions', 'exception-list'))

PLIST: key: 'Exceptions'
 value: String or cell-array with exceptions.

EXAMPLES: result = eq(c1,c2, 'name', 'created')
 result = eq(c1,c2, 'ao/name')
 result = eq(c1,c2, plist('Exceptions', 'name'))
 result = eq(c1,c2, plist('Exceptions', {'name', 'created'}))

INPUTS: c1,c2 - input pzmodel objects
 exc_list - exception list
 List of properties which are not checked.

OUTPUTS: If the two pzmodel objects are considered equal, result == 1,

VERSION: \$Id: eq.m,v 1.7 2008/02/18 14:41:13 ingo Exp \$

HISTORY: 29-08-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

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

ne

PURPOSE

NE overloads the `~=` operator for pzmodel objects.

SYNOPSIS

function result = ne(p1, p2, varargin)

DESCRIPTION

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

eq

PURPOSE

EQ overloads the == operator for specfin objects.

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)
 11.4801 0 0 11.48. /roproperty1'. /roproperty2')
 11.4801 0 0 11.48. class//roproperty'. class//roproperty')
11.4801 0 0 11.48. cl plist('Exceptions'. exception-list'))

PLIST: key: 'Exceptions'
 value: String or cell-7rray with exceptions.

INPUTS: 48. - input specwin objects

OUTPUTS: If the two specwin objects are considered equal,1.4801 0 = 0,
 otherwise,1.4801 0 = 1.

VERSION: \$Id: ne.m,v 1.4 2008/02/18 15:12:58 ingo Exp \$

HISTORY: 29-08-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

- [char](#) CHAR convert a specwin object into a string.
- [eq](#) EQ overloads the == operator for specfin objects.

This function is called by:

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

ne

PURPOSE

NE overloads the ~= operator for timeformat objects.

SYNOPSIS

function result = ne(varargin)

DESCRIPTION

NE overloads the ~= operator for timeformat objects.

%%%

DESCRIPTION: NE overloads the ~= operator for timeformat objects.

CALL: result = ne(tf1,tf2)
 result = ne(tf1,tf2, exc_list)
 result = ne(tf1,tf2, 'property1', 'property2')

'Excelt tf1,tf2, exc_list)tf - input the ~= operator fo'class/prOUTPUTSbjectsIfIPTIO

ne

PURPOSE

NE overloads the ~= operator for timespan objects.

SYNOPSIS

function result = ne(ts1, ts2, varargin)

DESCRIPTION

NE overloads the ~= operator for timespan objects.

%%%

DESCRIPTION: NE overloads the ~= operator for timespan objects.

CALL: result = ne(ts1,ts2)
 result = ne(ts1,ts2, exc_list)
 result = ne(ts1,ts2, 'property1', 'property2')
 result = ne(ts1,ts2, 'class/property', 'class/property')
 result = ne(ts1,ts2, plist('Exceptions', 'exception-list'))

PLIST: key: 'Exceptions'
 value: String or cell-array with exceptions.

INPUTS: ts1,ts2 - input timespan objects

OUTPUTS: If the two timespan objects are considered equal, result == 0,
 otherwise, result == 1.

VERSION: \$Id: ne.m,v 1.4 2005 OUTPUT9 3333jects

```
0001 function result = ne\(ts1, ts2, varargin\)  
0002 % NE overloads the ~= operator for timespan objects.
```


ne

PURPOSE

NE overloads the ~= operator for tsdata objects.

SYNOPSIS

function result = ne(c1, c2, varargin)

DESCRIPTION

NE overloads the ~= operator for tsdata objects.

%%%

DESCRIPTION: NE overloads the ~= operator for tsdata objects.

CALL: result = ne(c1,c2)
 result = ne(c1,c2, 'property1', 'property2')
 result = ne(c1,c2, 'class/property', 'class/property')
 result = ne(c1,c2, plist('Exceptions', 'exception-list'))

PLIST: key: 'Exceptions'
 value: String or cell-array with exceptions.

INPUTS: c1,c2 - input tsdata objects

OUTPUTS: If the two tsdata objects are considered equal, result == 0,
 otherwise, result == 1.

VERSION: \$Id: ne.m,v 1.4 2008/02/18 15:12:58 ingo Exp \$

HISTORY: 29-08-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function c7lls:

```
0001 function result = ne\(c1, c2, varargin\)  
0002 % NE overloads the ~= operator for tsdata objects.  
0003 %
```


[illegible]

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)

This function is called by:


```
0002 % NE overloads the ~= operator for xyzdata objects.
```

0003 %

0004 %%%%%%%%%%

0005 %

```
0006 % DESCRIPTION: NE overloads the ~= operator for xyzdata objects.003/RelatCj TCAL
```



```

0045     elseif strcmp(in, 'Version')
0046         b = VERSION;
0047         return
0048     elseif strcmp(in, 'Category')
0049         b = CATEGORY;
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
0050 r in_name{ en+1}i = inpu_namp(i));
0093 end
0604
061 [a's, p,= i vas]i =cCollec_ inpus( vanargi,r in_name));
0624
063 Top= '^';
0654
0565[a1,a2,do]i =aoooperCat( vanargi,rop));
0664
057 %----- creCat outnputA0
058 d
0695 %make a new hlisgor objlecd
070 hi =hlisgor(ALGONAME,= VERSIOs, plis(),5[a1.hlis a2.hlis]));
071 hi =r

```

```

0047     elseif strcmp(in, 'Category')
0048         b = CATEGORY;
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 op = '*';
0063
0064 [a1,a2,do] = aooperate(varargin, op);
0065
0066 %----- create output A0
0067
0068 % make a new history object
0069 h = history(ALGONAME, VERSION, plist(), [a1.hist a2.hist]);
0070 h = set(h, 'invars', invars);
0071
0072 % [et names for output
0073 if isempty(char(invars{1}))
0074     n1 = a1.name;
0075 else
0076     n1 = char(invars{1});
0077 end
0078 if isempty(char(invars{2}))
0079     n2 = a2.name;
0080 else
0081     n2 = char(invars{2});
0082 end
0083
0084 % make output analysis object
0085 b = ao(do, h);
0086 b = setnh(b, 'name', sprintf('%s %s %s', n1, op, n2))o 0i4'h o -ID 11 >,t9 %% G3
0086 b =

```


This function is called by:


```
0047     elseif strcmp(in, 'Category')
0048         b 801 0 0 11.4801 8.2001 809.4011 Tm (004.GORY; 11.T*01 809.4011 Treturn 1
```



```
0045         b = VERSION;
0046         return
0047     elseif strcmp(in, 'Category')
0048         b = CATEGORY;
0049 endfunction ofsieturn
```


0001

0057


```
0050 t2 = varargin{2};  
0051
```



```

0011 %
0012 % VERSION:          $Id: get_xy_values.m,v 1.5 2008/02/12 19:54:05 hewc31=Son Exp $
0013 %
0014 % HISTORY:          25-05-2007 Diepholz
0015 %                    Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 VERSION = '$Id: get_xy_values.m,v 1.5 2008/02/12 19:54:05 hewc=Son Exp $';
0020 CATEGORY = 'Internal';
0021
0022 % Check if this is a call for parameters
0023 if nargin == 2
0024     if isa(data, 'cdata') && ischar(pl)
0025         in = char(pl);
0026         if strcmp(in, 'Params')
0027             varargout{1} = plist;
0028             return
0029         elseif strcmp(in, 'Version')
0030             varargout{1} = VERSION;
0031             return
0032         elseif strcmp(in, 'Category')
0033             varargout{1} = CATEGORY;
0034             return
0035         end
0036     end
0037 end
0038
0039 x = [];
0040 y = [];
0041
0042 %% Set outputs
0043 if nargout == 1
0044     varargout{1} = data.y;
0045
0046 elseif nargout == 2
0047     varargout{1} = data.x;
0048     varargout{2} = data.y;
0049
0050 else
0051     error('### Incorrect output arguments.');
```

[Home](#) > [classes](#) > [@cdata](#) > set_xy_axis.m

set_xy_axis

PURPOSE

[Home](#) > [classes](#) > [@cdata](#) > single_operation.m

single_operation

PURPOSE


```

0043 if nargin == 2
0044     if isa(varargin{1}, 'cdata') && ischar(varargin{2})
0045         in = char(varargin{2});
0046         if strcmp(in, 'Params')
0047             varargout{1} = getDefaultPL();
0048             return
0049         elseif strcmp(in, 'Version')
0050             varargout{1} = VERSION;
0051             return
0052         elseif strcmp(in, 'Category')
0053             varargout{1} = CATEGORY;
0054             return
0055         end
0056     end
0057 end
0058
0059 % Only the call from an analysis object is allowed.
0060 if (nargin == 3)
0061
0062     data = varargin {1};
0063     op    = varargin {2};
0064     pl    = varargin {3};
0065     a     = varargin {end};
0066     y     = data.y;
0067
0068 %% Verify input variables and set default parameter
0069     if ~isa(data, 'cdata')
0070         error ('### cdata/single_operation: The first input must be a cdata.');
```

```

0071     end
0072     if ~ischar(op)
0073         error ('### cdata/single_operation: The second input must be a string.');
```

```

0074     end
0075     if ~isa(plist, 'plist')
0076         error ('### cdata/single_operation: The third input must be a plist.');
```

```

0077     end
0078
0079 %% Set default parameter list
0080     pl_default = getDefaultPL();
0081
0082 %% Combine the handover param-list and the default param-list
0083     if ~isempty(pl)
0084         pl = combine(pl);
0085     else
0086         pl = combine(pl_default);
0087     end
0088
0089     option      = find(pl, 'option');
0090     dim          = find(pl, 'dim');
0091     weight_vector = find(pl, 'W');
```



```
0013 %                               Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017
0018 % Check if this is a call for parameters
0019 if ischar(S)
0020     in = char(S);
0021     if strcmp(in, 'Params')
0022         B = plist();
0023         return
0024     elseif strcmp(in, 'Version')
0025         VERSION = '$Id: subsref.m,v 1.9 2008/02/22 22:26:40 hewitson Exp $';
0026         B = VERSION;
0027         return
0028     elseif strcmp(in, 'Category')
0029         CATEGORY = 'Internal';
0030         B = CATEGORY;
0031         return
0032     end
0033 end
0034
0035 B = builtin('subsref', A, S);
0036
```

[Home](#) > _____


```

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 1.5 2008/02/12 22:11:21 hewitson Exp $
0015 %
0016 % HISTORY:        31-01-2007 M Hewitson
0017 %                  Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 VERSION = '$Id: reshapeF.m,v 1.5 2008/02/12 22:11:21 hewitson Exp $';
0022 CATEGORY = 'Internal';
0023
0024 if nargin == 2 && isa(fsd, 'fsdata') && ischar(varargin{1})
0025     if strcmp(varargin{1}, 'Params')
0026         fsd = plist();
0027         return
0028     elseif strcmp(varargin{1}, 'Version')
0029         fsd = VERSION;
0030         return
0031     elseif strcmp(varargin{1}, 'Category')
0032         fsd = CATEGORY;
0033         return
0034     end
0035 end
0036
0037 if size(fsd.x, 1) ~= size(fsd.y,1)
0038     fsd.x = [fsd.x].';
0039 end
0040 if size(fsd.x, 1) ~= size(fsd.y,1)
0041     error('### frequency and data vectors are not the same length.');
```


[Home](#) > [classes](#) > [@fsdata](#) > single_operation.m

single_operation

PURPOSE 


```

0095 option      = find(pl, 'option');
0096 dim          = find(pl, 'dim');
0097 weight_vector = find(pl, 'w');
0098 do_xdata      = find(pl, 'xdata');
0099 do_ydata      = find(pl, 'ydata');
0100
0101 %% Check the <char> option if it begins and ends with a quote '
0102 %% If not then add the quote.
0103 if ischar(option) && ~isempty(option)
0104     if ~strcmp(option(1), '')
0105         option = [' ' option ' '];
0106 pl, '1091ttion la');
0107
0108 %%ntof iquotl, 'ydata'1 add the numerg e.

```



```

0151      %% e.g.: x_new = '[ x_new1 x_new2 ]'
0152      x_new = '[';
0153      for i = 1:nargout-1
0154          new_var_name = genvarname (sprintf('x_new%d',i));
0155          eval ([new_var_name '= 0;']);
0156
0157          x_new = strcat(x_new, sprintf (' x_new%d',i));
0158t 6nd);
97      x_new = str cat(x_ne'ew2i));

```

```

sprintfdata_argnew%d',i));
[new_var_name data; 0;']);

```

```
do_ydata)56
```

```
operat)56
```



```
0263         eval (sprintf('data_out%d.yunits = '%s(%)';',i,op,data.yunits));
0264     end
0265     %% t data (x-axis)
0266     if ~isempty(do_xdata)
0267         eval (sprintf('data_out%d.x = x_new%d;',i,i));
0268         eval (sprintf('data_out%d.name = '%s(%)';',i,op,data.name));
```



```
0011 % VERSION:      $Id: subsasgn.m,v 1.6 2008/02/25 18:06:31 ingo Exp $
0012 %
0013 % HISTORY:       31-01-07 M Hewitson
0014 %                Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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')
0023taA    iplist;s')

iVamerearams')

iCategoryrams')0023tendtion((S);)Tj 210.8 -1.343 Td39tendtion'Id: subs', saS, B((S);)
```

[Home](#) > [classes](#) > [@fsdata](#) > subsref.m

subsref

PURPOSE

SUBSREF Define field name indexing for fsdata objects.

SYNOPSIS

```
0013 %                               Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 % Check if this is a call for parameters
0018 if ischar(S)
0019     in = char(S);
0020     if strcmp(in, 'Params')
0021         B = plist;
0022         return
0023     elseif strcmp(in, 'Version')
0024         VERSION = '$Id: subsref.m,v 1.15 2008/02/22 22:26:40 hewitson Exp $';
0025         B = VERSION;
0026         return
0027     elseif strcmp(in, 'Category')
0028         CATEGORY = 'Internal';
0029         B = CATEGORY;
0030         return
0031     end
0032 end
0033
0034 B = builtin('subsref', A, S);
```

```
0057     h.n = a;
0058 end
0059
0060 % set my parent node
0061 n8n/o08ac9
002 if1 n<0
0037     hpn8n/8ac
0047     n8n/[n/8a]c
0057     pl8n/o08listc
0067     nNode(a)hpn88888n/8ac
0077     nNode(a)hinvars8n/o0invarsc
0087     nNode(a)hn888888n/o0ac9
0097     nNode(a)hnames88n/o0aamec
0707     nNode(a)hparams8n/s
```


SOURCE CODE

```
0001 function [varargout] = getNodes_plot(varargin)
0002 % GETNODES_PLOT converts a history object to a nodes structure suitable for
```


[Home](#) > [classes](#) > [@history](#) > subsasgn.m

subsasgn

PURPOSE 

```
0022 end %
35 end %
0011 % VERSION: $Id: subsasgn.m,v 1.5 2008/02/25 18:06:31 ingo Exp $
37 if isempty(A) %
0012 end %
49 %
0013 % HISTORYVERSI.0-00-07 M Hewitson %
401A = builtin('subsas', A, S, B);
0413 mC 0 m 0 -0.64 l -578.6 -6 Creinat 18% 82 l - 82 - 82 l - 82 - 82 l 801mGS, B);79nq 0 18 595 806 re W* n BT /TT1 1.
0512 %
06 %%%%%%%%%%% %
07
0813 Check if this is a call for pararimers
09 if narPag == 2 %
20 if isa(A, 'history') && ischar(S) %
```


subsref

PURPOSE

SUBSREF Define field name indexing for mfir objects.

SYNOPSIS

function B = subsref(A, S)

DESCRIPTION

SUBSREF Define field name indexing for mfir objects.

%%%

DESCRIPTION: SUBSREF Define field name indexing for mfir objects.

EXAMPLES: All possible accesses are possible.

VERSION: \$Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp \$

HISTORY: 31-01-07 M Hewitson
Creation

CROSS-REFERENCE INFORMATION

This function calls:

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

```

0013 %                               Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 % Check if this is a call for parameters
0018 if ischar(S)
0019     in = char(S);
0020     if strcmp(in, 'Params')
0021         B = plist;
0022         return
0023     elseif strcmp(in, 'Version')
0024         VERSION = '$Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $';
0025         B = VERSION;
0026         return
0027     elseif strcmp(in, 'Category')
0028         CATEGORY = 'Internal';
0029         B = CATEGORY;
0030         return
0031     end
0032 end
0033
0034 B = builtin('subsref', A, S);

```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

0012	%	HISTORY:	31-01-07 M Hewitson
0013	%		Creation
0014	%		
0015	%	%	%


```

0013 %                               Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 % Check if this is a call for parameters
0018 if ischar(S)
0019     in = char(S);
0020     if strcmp(in, 'Params')
0021         B = plist;
0022         return
0023     elseif strcmp(in, 'Version')
0024         VERSION = '$Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $';
0025         B = VERSION;
0026         return
0027     elseif strcmp(in, 'Category')
0028         CATEGORY = 'Internal';
0029         B = CATEGORY;
0030         return
0031     end
0032 end
0033
0034 B = builtin('subsref', A, S);

```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

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

subsasgn

PURPOSE 

```
0012 % HISTORY:      31-01-07 M Hewitson
0013 %              Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 % Check if this is a call for parameters
0018 if ischar(S)
0019     in = char(S);
0020     if strcmp(in, 'Params')
0021         A = plist;
0022         return
0023     elseif strcmp(in, 'Version')
0024         VERSION = '$Id: subsasgn.m,v 1.7 2008/02/25 18:06:31 ingo Exp $';
0025         A = VERSION;
0026         return
0027     elseif strcmp(in, 'Category')
0028         CATEGORY = 'Internal';
0029         A = CATEGORY;
0030         return
0031     end
0032 end
0033
0034 if isempty(A)
```

[Home](#) > [classes](#) >

subsasgn

PURPOSE

SUBSASGN define index assignment for parameter list properties.

SYNOPSIS

function A = subsasgn(A, S, B)

DESCRIPTION

SUBSASGN define index assignment for parameter list properties.

%%%

DESCRIPTION: SUBSASGN define index assignment for parameter list properties.

EXAMPLES: All possible assignment are possible.

VERSION: \$Id: subsasgn.m,v 1.9 2008/02/25 18:06:31 ingo Exp \$

HISTORY: 31-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

```

0011 % VERSION:      $Id: subsasgn.m,v 1.9 2008/02/25 18:06:31 ingo Exp $
0012 %
0013 % HISTORY:       31-01-07 M Hewitson
0014 %                Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 %
0018 % Check if this is a para's parameters
0019 if nargin == 2
0020     if isa(A, 'plist') && ischar(S)
0021         in =

```

```

041 An =built in(' subsasg', (A,S, BS; )Tj T* (042 )Tj ET E

```

```

040 n

```

```

039 endn

```


0060 % END

subsasgn

SUBSASGN define index assignment for pole properties.

SYNOPSIS

function A = subsasgn(A, S, B)

DESCRIPTION

SUBSASGN define index assignment for pole properties.

%%%

DESCRIPTION: SUBSASGN define index assignment for pole properties.

EXAMPLES: All possible assignment are possible.

VERSION: \$Id: subsasgn.m,v 1.5 2008/02/25 18:06:31 ingo Exp \$

HISTORY: 31-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

[Home](#) > [classes](#) > [@provenance](#) > subsasgn.m

[Home](#) > [classes](#) > [@provenance](#) > subsref.m

subsref

PURPOSE

```
0013 %                               Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 % Check if this is a call for parameters
0018 if ischar(S)
0019     in = char(S);
0020     if strcmp(in, 'Params')
0021         B = plist;
0022         return
0023     elseif strcmp(in, 'Version')
0024         VERSION = '$Id: subsref.m,v 1.5 2008/02/22 22:26:41 hewitson Exp $';
0025         B = VERSION;
0026         return
0027     elseif strcmp(in, 'Category')
0028         CATEGORY = 'Internal';
0029         B = CATEGORY;
0030         return
0031     end
0032 end
0033
0034 B = builtin('subsref', A, S);
0035
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

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

getupperFreq

PURPOSE 

```

0009 % CALL:          f = getupperFreq(pzm);
0010 %
0011 % VERSION:       $Id: getupperFreq.m,v 1.4 2008/02/12 20:07:37 hewitson Exp $
0012 %
0013 % HISTORY:        04-04-2007 M Hewitson
0014 %                  Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 VERSION = '$Id: getupperFreq.m,v 1.4 2008/02/12 20:07:37 hewitson Exp $';
0019 CATEGORY = 'Internal';
0020
0021 % Check if this is a call for parameters
0022 if nargin == 2
0023     if isa(pzm, 'pzmodel') && ischar(varargin{1})
0024         in = char(varargin{1});
0025         if strcmp(in, 'Params')
0026             f = plist;
0027             return
0028         elseif strcmp(in, 'Version')
0029             f = VERSION;
0030             return
0031         elseif strcmp(in, 'Category')
0032             f = CATEGORY;
0033             return
0034         end
0035     end
0036 end
0037
0038 poles = get(pzm, 'poles');0032  t</MCn, 'Paramso=

```

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

subsasgn

PURPOSE

SUBSASGN define index assignment for pzmodel properties.

SYNOPSIS `S = subsasgn(A, 'S', 0, 'Am@5bT', '/TT1 1 Tf', 15.8956, 0, 0, 15.8956, 8.2001, 66, 'ties');`

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

subsref

PURPOSE

```
0013 %                               Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 % Check if this is a call for parameters
0018 if ischar(S)
0019     in = char(S);
0020     if strcmp(in, 'Params')
0021         B = plist;
0022         return
0023     elseif strcmp(in, 'Version')
0024         VERSION = '$Id: subsref.m,v 1.4 2008/02/22 22:26:40 hewitson Exp $';
0025         B = VERSION;
0026         return
0027     elseif strcmp(in, 'Category')
0028         CATEGORY = 'Internal';
0029         B = CATEGORY;
0030         return
0031     end
0032 end
0033
0034 B = builtin('subsref', A, S);
```

[Home](#) > [classes](#) > _____

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

subsref

PURPOSE 

```
0013 %                               Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 % Check if this is a call for parameters
0018 if ischar(S)
0019     in = char(S);0 eT29.0 -1343 Td* (0209    of trcmp(in, 'Pparas'S))Tj T* (021
        023    else of trcmp(in, 'Vtertio'S)

        027    else of trcmp(in, 'Category'S)

        034 Bn =builtin(' subsre', A, (S);0 e T* (035r)Tj ET EMC   HRP <</MCID
```


0012	%	HISTORY:	31-01-07 M Hewitson
0013	%		Creation
0014	%		
0015	%	%	%

[Home](#) > _____

[Home](#) > [classes](#) > [@timeformat](#) > subsasgn.m

subsasgn

PURPOSE

SUBSASGN define index assignment for timeformat properties.

SYNOPSIS

function A = subsasgn(A, S, B)

DESCRIPTION

[Home](#) > [classes](#) > [@timeformat](#) > subsref.m

subsref

PURPOSE

[Home](#)

```
0012 % HISTORY:      31-01-07 M Hewitson
0013 %              Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 % Check if this is a call for parameters
0018 if nargin == 2
0019     if isa(A, 'timespan') && ischar(S)
0020         in = _____
```



```
0013 %                               Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 % Check if this is a call for parameters
0018 if ischar(S)
0019     in =
```

[Home](#) > [classes](#) > [@tsdata](#) > get_xy_values.m

get_xy_values

PURPOSE

SYNOPSIS

function varargout = get_xy_values(data, pl) DescrES.91:xO/c2ET EMC /I <</MCID 12 >>B6C 1 0 0 RG 1 w 82

0012 %
0013 % HISTORY: 21-06-2007 Diepholz
0014 % Creation

[Home](#) > [classes](#) > [@tsdata](#) > single_operation.m

single_operation

PURPOSE

SINGLE_OPERATION /TT1 1 Tf 11.4Aelements specify ET Eor overload for EMC objects./H2 <</MCID 7 >

This function calls:

 θ

```
0038 %
0039 % HISTORY: 07-05-2007 Diepholz
0040 %          Creation.
0041 %
0042 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0043
vararBBo{1}, 'tsdata') && ischar(vararBBo{2}) TD (0039 % HIST050ion.)Bo =044 VCID <</M
```

```
0095    option          = find(pl, 'option');
```

```

0151      %% e.g.: x_new = '[ x_new1 x_new2 ]'
0152      x_new = '[';
0153      for i = 1:nargout-1
0154          new_var_name = genvarname (sprintf('x_new%d',i));
0155          eval ([new_var_name '= 0;']);
0156
0157          x_new = strcat(x_new, sprintf (' x_new%d',i));
0158      end
0159      x_new = strcat (x_new, ' ]');
0160
0161  %% %% Create the output variables
0162  %% e.g.: data_out1 = data;
0163  for i = 1:nargout-1
0164      new_var_name = genvarname (sprintf('data_out%d',i));
0165      eval ([new_var_name '= data;']);
0166  end
0167
0168  %% Execute the command for y (y-axis)
0169  if ~isempty(do_ydata)
0170
0171      try
0172          %% Execute with a option
0173          %% >> operation(y,'option');
0174          if ~isempty(option)
0175              cmd_line = sprintf('%s = %s(y,%s);',y_new, op, option);
0176              eval (cmd_line);
0177
0178          %% Execute with a weight vector and dimension
0179          %% >> operation(y,W,dim);
0180          elseif ~isempty(dim) && ~isempty(weight_t ,t0aaaaaaaaaaaaas1.2 e perati017
0179 2dim) && ~i,y_new, op, option);0176          8          %% >>(cmd_line);)Tj T*8          %%

```



```
0012 % HISTORY:      31-01-07 M Hewitson
0013 %              Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 % Check if this is a call for parameters
0018 if nargin == 2
0019     if isa(A, 'tsdata') && ischar(S)
0020         in = char(S);
0021         if strcmp(in, 'Params')
0022             A = plist;
0023             return
0024         elseif strcmp(in, 'Version')
0025             VERSION = '$Id: subsasgn.m,v 1.6 2008/02/25 18:06:32 ingo Exp $';
0026             A = VERSION;
0027             return
0028         elseif strcmp(in, 'Category')
0029             CATEGORY = 'Internal';
0030             A = CATEGORY;
0031             return
0032         end
0033     end
0034 end
0035
0036 if isempty(A)
0037     A = tsdata();
0038 end
0039
0040 A = builtin ('subsasgn', A, S, B);
0041
```

[Home](#) > [classes](#)

[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 1.4 2008/02/13 12:57:47 mauro Exp \$

HISTORY: 21-06-2007 Diepholz
 Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

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

This function is called by:

SOURCE CODE

```
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 2008/02/13 12:57:47 mauro Exp $
```

0012 %
0013 % HISTORY: 21-06-2007 Diepholz
0014 % Creation


```

0207         end
0208
0209     end
0210
0211 %% Execute the command for the ydata
0212 if ~isempty(do_ydata)
0213
0214     try
0215         %% Execute with a option or a dimension(option)
0216         cmd_line = sprintf('%s = %s(y,%s);',y_new, op, option);
0217         eval (cmd_line);
0218
0219         %% Execute with a weight vector and dimension
0220
0221         cmd21 _ %%d_line = sprintf(' %% if25ption);
0222
0223         %% Execute with a op28 dimension
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

```


SUBSASGN define index .89fgnment for xydata properties.

function A = subsasgn(A, S, B)

SUBSASGN define index .89fgnment for xydata properties.

[illegible]

DESCRIPTION: SUBSASGN define index .89fgnment for xydata properties.

EXAMPLES: All po89fble .89fgnment are po89fble.

VERSION: \$Id: subsasgn.m,v 1.6 2008/02/25 18:06:32 ingo Exp \$

HISTORY: 31-01-07 M Hewitson
Creation

[illegible]

This function calls:

- o [char](#) CHAR convert a param object into a string.
- o [xydata](#) XYDATA X-Y data object class constructor.

This function is called by:

[illegible]

subsref

PURPOSE

SUBSREF Define field name indexing for xydata objects.

SYNOPSIS

function B = subsref(A, S)

DESCRIPTION

SUBSREF Define field name indexing for xydata objects.

%%%

DESCRIPTION: SUBSREF Define field name indexing for xydata objects.

EXAMPLES: All possible accesses are possible.

VERSION: \$Id: subsref.m,v 1.8 2008/02/22 22:26:40 hewitson Exp \$

HISTORY: 31-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION

This function calls:

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

This function is called by:

SOURCE CODE

```
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 1.8 2008/02/22 22:26:40 hewitson Exp $
0011 %
0012 % HISTORY: 31-01-07 M Hewitson
```



```
0113
0114 elseif nargout == 3
0115     varargout{1} = x;
0116     varargout{2} = y;
0117     varargout{3} = z;
0118 else
0119     error('### Incorrect output arguments. ');
0120 end
0121
0122 % END
```

[Home](#) > [classes](#) > [@xyzdata](#) > single_operation.m

single_operation

PURPOSE

SINGLE_OPERATION< implements specified operator overload for xyzdata objects.

SYNOPSIS

```
>> operation(data,w,dim)
>> operation(data,dim)
>> operation(data,w)
>> operation(data)
```

HISTORY: 24-12-2007 Diepholz
Creation.

%%

CROSS-REFERENCE INFORMATION


```
0091     error ('### xyzdata/single_operation: The third input must be a plist.');
```

0092 end

0093

```
0094 %% Set default parameter list (0094    %%5    pl_default = )Tj ET EMC  /Link <</M
```



```
Have a look to the param list', cmd_line);
```

```
0260         end
```

```
0261
```

```
0262     end
```

```
0263
```

```
026_ope63 809.5682 Tm (Have a look to theg BTle_operation)THave ae/]om eval is in x_ne
```



```
0016
0017 % Check if this is a call for parameters
0018 if ischar(S)
0019     in = char(S);
0020     if strcmp(in, 'Params')
0021         B = plist;
0022         return
0023     elseif strcmp(in, 'Version')
0024         VERSION = '$Id: subsref.m,v 1.3 2008/02/22 22:26:40 hewitson Exp $';
0025         B = VERSION;
0026         return
0027     elseif strcmp(in, 'Category')
0028         CATEGORY = 'Internal';
0029         B = CATEGORY;
0030         return
0031     end
0032 end
0033
0034 B = builtin('subsref', A, S);
0035
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003


```

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 1.5 2008/02/12 19:54:16 hewitson Exp $
0018 %
0019 % HISTORY:      03-04-2007 Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 VERSION = '$Id: cz2iir.m,v 1.5 2008/02/12 19:54:16 hewitson Exp $';
0025 CATEGORY = 'Internal';
0026
0027 %%%% 'Params' && 'Version' Call %%%%
0028 if nargin == 2
0029     if isa(z, 'zero') && ischar(fs)
0030         in = fs;
0031         if strcmp(in, 'Params')
0032             a = plist();
0033             return
0034         elseif strcmp(in, 'Version')
0035             a = VERSION;
0036             return
0037         elseif strcmp(in, 'Category')
0038             a = CATEGORY;
0039             return
0040         end
0041     end
0042 end
0043
0044 f0 = get

```

Generated on Tue 25-Mar-2008 23:00:00 by

[Home](#) > [classes](#) > [@zero](#) > rz2iir.m

rz2iir

PURPOSE

RZ2IIR Return a,b IIR filter coefficients for a real zero designed using the bilinear transform.

SYNOPSIS


```

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 1.6 2008/02/12 19:54:16 hewitst ae260016 %uld onl
0005 %
0006 28 if nar8 8 == 205 %

0006 44 f0 =r/MCIDTID 0 >/Link q 0 18 51 i /Relati1 RG 0.551 wre 001 809 77.081 25

```


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

subsref

PURPOSE

SUBSREF Define field name indexing for zero objects.

SYNOPSIS


```
0012 % M Hewitson 26-02-08
0013 %
0014 % $Id: ltpda_disp.m,v 1.1 2008/02/26 09:30:01 hewitson Exp $
0015
0016
0017 ALGONAME = mfilename;
0018 VERSION   = '$Id: ltpda_disp.m,v 1.1 2008/02/26 09:30:01 hewitson Exp $';
0019 CATEGORY = 'Internal';
0020
0021 % Check if this is a call for parameters
0022 if nargin == 1
0023     if ischar(varargin{1})
0024         in = char(varargin{1});
0025         if strcmp(in, 'Params')
0026             varargout{1} = plist;
0027             return
0028         elseif strcmp(in, 'Version')
0029             varargout{1} = VERSION;
0030             return
0031         elseif strcmp(in, 'Category')
0032             varargout{1} = CATEGORY;
0033             return
0034         end
0035     end
0036 end
0037
000 l S Q   n C   010 0 ri 2TT0 1 Tf 12.4925 0 0 12.49234.4379 250.0805137 Tm2ht
```


HISTORY: 10-05-2007 M Hewitson
Creation

[illegible]

CROSS-REFERENCE INFORMATION

```
0030 %    >> info  = ltpda_dbquery(conn, 'transactios', 'user_id=3');
0031 %    >> info  = ltpda_dbquery(conn, 'transactios', 'obj_id=56');
0032 %
0033 %    >> info  = ltpda_dbquery(conn)
0034 %
0035 %    The 'info' structure will contain a list of the tables in the database.
0036 %
0037 %    >> info  = ltpda_dbquery(conn, query)
0038 %
0039 %    The 'info' structure will contain a list of records resulting from the SQL
```

```

0086
0087 else
0088     error('### Incorrect inputs.');
```

```

0089 end
0090
0091 varargout{1} = info;
0092
0093 %-----
0094 % Get table list
0095 function info = getTableList\(conn\)
0096
0097 % open a connection
0098 try
0099     curs = exec(conn, 'show tables');
0100     curs = fetch(curs);
0101     info = curs.Data;
0102     close(curs);
0103 catch
0104     error('### Failed to get table list. Server returned: %s', curs.Message);
0105 end
0106
0107 %-----
0108 % Get field list
0109 function info = getFieldList\(conn, table\)
0110
0111 try
0112     curs = exec(conn, sprintf('describe %s', table));
0113     curs = fetch(curs);
0114     info = curs.Data;
0115     close(curs);
0116 catch
0117     error('### Failed to get field list. Server returned: %s', curs.Message);
0118 end
0119
0120 %-----
0121 % Get field list
0122 function info = simpleQ ET \(conn, q\)
0123
0124 % open a connection
0125
0126 try
0127     curs = exec(conn, sprintf('%s', q));
0128     curs = fetch(curs);
0129     info = curs.Data;
```


[Home](#) >

CROSS-REFERENCE INFORMATION

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

```
>> category = ltpda_ltf_plan('Category')
```

REFERENCE: "lpsd revisited: ltf" / S2-AEI-TN-3052
2008/02/07 V1.1
G Heinzel

M Hewitson 19-02-08

\$Id: ltpda_ltf_plan.m,v 1.1 2008/02/19 09:11:54 hewitson Exp \$

%%

CROSS-REFERENCE INFORMATION

```

0021 %           Kdes    - The desired number of averages.
0022 %
0023 %   Outputs:
0024 %           Each output is a vector, one value per frequency:
0025 %
0026 %           f        - the frequency
0027 %           r        - frequency resolution (Hz)
0028 %           b        - bin number
0029 %           L        - segment lengths
0030 %           K        - number of averages
0031 %
0032 %   Parameter list:
0033 %
0034 % The following call returns a parameter list object that contains the
0035 % default parameter values:
0036 %
0037 % >> pl = ltpda_ltf_plan('Params')
0038 %
0039 % The following call returns a string that contains the routine CVS version:
0040 %
0041 % >> version = ltpda_ltf_plan('Version')
0042 %
0043 % The following call returns a string that contains the routine category:
0044 %
0045 % >> category = ltpda_ltf_plan('Category')
0046 %
0047 %
0048 % REFERENCE:  "lpsd revisited: ltf" / S2-AEI-TN-3052
0049 %             2008/02/07  V1.1
0050 %             G Heinzel
0051 %
0052 %
0053 % M Hewitson 19-02-08
0054 %
0055 % $Id: ltpda_ltf_plan.m,v 1.1 2008/02/19 09:11:54 hewitson Exp $
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %
0059 %           q        - a vector of start indices for the segments
0060 %
0061 %
0062 % VERSION  = '$Id: ltpda_ltf_plan.m,v 1.1 2008/02/19 09:11:54 hewitson Exp $';
0063 % CATEGORY = 'Internal';
0064 %
0065 %
0066 %% Check if this is a call for parameters, the CVS version string
0067 % or the function category
0068 if nargin == 1 && ischar(varargin{1})
0069     in = char(varargin{1});
0070     if strcmp(in, 'Params')

```


0192 % END

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003


```

0109     nsecs = 0;
0110     start = 1;
0111     while start + l - 1 <= nx
0112         idxs(1+nsecs,:) = start:start+lm1;
0113         start = floor(start + l*Xolah);
0114         nsecs = nsecs + 1;
0115     end
0116     Sxy = zeros(nsecs,1);
0117     i2 = zeros(nsecs,1);
0118     xi = zeros(l,1);
0119     xo = zeros(l,1);
0120
0121     %----- Process the channels
0122
0123     % Lo]/Tover input channels
0124     for j=1:nc
0125         % lo]/Tover output channels
0126         for k=1:nc
0127             % compute lower triangle of matrix
0128             if j<k
0129
0130                 % do segments
0131                 Sxx = 0;
0132                 Syy = 0;
0133                 Sxy(:) = 0;
0134                 i2(:) = 0;
0135
0136                 % Lo]/Tover segments
0137                 for s=1:nsecs
0138                     % get segments
0139                     xi(:) = X(idxs(s,:),j);
0140                     xo(:) = X(idxs(s,:),k);
0141
0142                     % detrend segments
0143                     eval(detrender);
0144
0145                     % compute DFT
0146                     ti = C*xi;
0147                     to = C*xo;
0148                     cto = conj(to);
0149                     Sxy(s) = cto*ti;
0150                     Sxxxy(s)/W+ *= conjier);
0150                     Syy (s Sy+ o*= c);
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

```


SOURCE CODE

```
0001 function varargout = ltpda\_loglike\(varargin\)
0002 %Log likelihood criterion
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: ltpda_loglike computes the "crunched" log likelihood
0006 % criterion of two time series.
0007 %
```



```
0108     for j=1:n
0109         a(j,i) = b(j, 2*i) + b(j, 2*i-1);
0110     end
0111 end
0112 end
0113
0114 function p = norm2lag\(y,S,lag\)
0115 % dot product with weight S and time lag
0116 [h,l] = size(y);
0117 p = zeros(h,h);
0118j 0 -1 = 1:h
0119     for k = 1:h
0120         x1 = y(k,1+lag:l);
```

[Home](#) > [m](#) >

This function is called by:


```
0102     Width(j_meta + 2*Nss_meta ) = InputSizes_meta(j_meta);
0103 end
0104 % Filling the width vectors for Ox and Oy
0105 for i_meta=1:Nout_meta
0106     for j_meta=1:Nss_meta
```


[Home](#)

***** THERE ARE NO DEFAULT PARAMETERS *****

VERSION: \$Id: ltpda_ss_iir2ss.m,v 1.3 2008/03/11 16:52:56 adrien Exp \$

ltpda_ss_make

PURPOSE 


```
0086     elseif isequal(class(varargin{i}), 'pzmodel')
0087         pzm = varargin{i};
0088         display('not fully implemented yet');
0089         subsys = _____
```


with user defined values

0023 % 'SET_PARAM_VALUES' Array of parameter means for numerical substitutions

0024 % 'SET_PARAM_SIGMAS' Array of parameter standard deviation for numerical substitutions (unit may be none!!!)

0025 % 'SET_PARAM_VALUES_ORDER' Array of 1 / 0 depending whether parameter means are effectively substituted or not

0026 % 'SET_PARAM_SIGMAS_ORDER' Array of 1 / 0 depending whether parameter means are effectively substituted or not

0027 %

0028 % 'JAC_PARAM_NAMES' same for parameters used for evaluating a jacobian

0029 % 'JAC_PARAM_ORDER'

0030 % 'HESS_PARAM_NAMES' same for parameters used for evaluating a hessian

0031 % 'HESS_PARAM_ORDER'


```
0183         end
0184     end
0185 %% building updated list for JacParam...
0186     if isequal(JacParamOrder,'ALL')
0187         JacParamNames = JacParamNamesIni;
0188     elseif isequal(JacParamOrder,'NO')
0189         JacParamNames = {};
0190     else
0191         JacParamNames = {};
0192         j=1;
0193         for i=1:length(JacParamNamesIni)
0194             if JacParamOrder(i)
0195                 JacParamNames{j} = JacParamNamesIni{i}; %#ok<AGROW>
0196                 j = j+1;
0197             end
0198         end
0199     end
0200 %% building updated list for HessParam...
0201     if isequal(HessParamOrder,'ALL')
0202         HessParamNames = HessParamNamesIni;
0203     elseif isequal(HessParamOrder,'NO')
0204         HessParamNames = {};
0205     else
0206         HessParamNames = {};
0207         j=1;
0208         for i=1:length(HessParamNamesIni)
0209             if HessParamOrder(i)
0210                 HessParamNames{j} = HessParamNamesIni{i}; %#ok<AGROW>
0211                 j = j+1;
```

```
0240         end
0241     for j=1:nHess
0242         if isequal(SystemParamNames{i},HessParamNames{j})
0243             InHess(i) = j;
0244         end
0245     end
0246 end
0247
0248 %% looking for non existing but modified parameter names
0249 if not(length(NotUsedParamNames)==sum(InNotUsed>0))
0250     msg = 'error with notusedparameter list because one does not exist
in system.';
0251     error(msg);
0252 elseif not(length(SetParamNames)==sum(InSet>0))
0253     msg = 'error with setparameter list because one does not exist in
system.';
0254     error(msg);
0255 elseif not(length(JacParamNames)==sum(InJac>0))
0256     msg = 'error with jacparameter list because one does not exist in
system.';
0257     error(msg);
0258 elseif not(length(nd
```


VERSION: \$Id: ltpda_ss_modify_time_setup.m,v 1.5 2008/03/11 16:52:56 adrien Exp \$

HISTORY: 08-02-2008 A Grynagier
24-01-2008 A Grynagier
Creation 16-01-2008 A Grynagier

to do : modify for noise shape filter case or HF inputs with aliasing

%%%

% standard calls for LTPDA function data

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

```
0034 % 'AMAT' cell array (contains A matrix)
0035 % 'BMATS' cell array (contains B matrices)
0036 % 'CMAT' cell array (contains C matrix)
0037 % 'DMATS' cell array (contains D matrices)
0038 % ***** THERE ARE NO DEFAULT PARAMETERS *****
0039 %
```


0145
0146
0147
0148
0149
0150
0151
0152
0153
0154
0155
0156
0157
NSubstep
0158
NSubstep
0159
0160
0161
0162
0163
0164
0165
0166
0167 %%


```
0024 % 'Timestep' is a real positive integer, set to zero in continuous case
0025 % 'PARAMNAMES' cell array of strings describing parameter variables
```


[Home](#) >


```
0114          'INPUTVARNAMES', SystemInputVarNames2, 'OUTPUTVARNAMES',  
SystemOutputVarNames2 ); %#ok<AGROW>  
0115      end  
0116 end  
0117 varargout = {System_out};  
0118 end
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003


```
0055      ao_in_list = ao_in_lists(i_ao_in);  
syst, 'DMATS'_in);
```

```
-se806s data)'Y4TD 63055o8168055  sts(TPUcell(t, 'NBINe(A,1_in);
```

```

0108
0109     %% generating random data
0110     for i=1:NBINPUTS
0111         if isequal(class(ao_in{i}), 'plist')
0112             sigma = find(ao_in{i}, 'SIGMA');
0113             mean = find(ao_in{i}, 'MEAN');
0114             if sigma==0
0115                 ao_in{i} = ao(ones(nsamples, INPUTSIZES(i))*diag(mean));
0116             else
0117                 ao_in{i} = ao(ones(nsamples, INPUTSIZES(i))*diag(mean) +
0118 randn(nsamples, INPUTSIZES(i))*sigma);
0119             end
0120         end
0121     end
0122
0123     %% simulation first step
0124     x = zeros(nss, nsamples);
0125     y = zeros(size(C,1), nsamples);
0126     x(:,1) = Xini;
0127     IncrementY = zeros(size(C,1),1);
0128     for i=1:NBINPUTS
0129         if INPUTISUSED(i)
0130             IncrementY = IncrementY + D{i}* ao_in{i}.data.y(1,:)' ;
0131         end
0132     end
0133     y(:,1) = C* x(:,1) +IncrementY;
0134
0135     %% simulation loop
0136     for k = 2:nsamples
0137         IncrementX = zeros(nss,1);
0138         IncrementY = zeros(size(C,1),1);
0139         for i=1:NBINPUTS
0140             if INPUTISUSED(i)
0141                 IncrementX = IncrementX + B{i}*ao_in{i}.data.y(k,:)' ;
0142                 IncrementY = IncrementY + D{i}*ao_in{i}.data.y(k,:)' ;
0143             end
0144         end
0145         x(:,k) = A* x(:,k-1) + IncrementX ;
0146         y(:,k) = C* x(:,k) + IncrementY ;
0147     end
0148
0149     %% data obj      fo;

```

```
0164         % hi_old = ao_in.hist ;
0165         hi_new = history('simulate_discrete','version', syst);
0166         % hi = [hi_old hi_new];
0167         % h = history(hi);
0168         h = history(hi_new);
0169         ao_out_loc = ao(ts,h);
0170         ao_out_loc.name = ALGONAME;
0171         ao_out(i_syst , i_ao_in) = ao_out_loc;
0172     end
0173 end
0174 %% construct output analysis object
0175 varargout = {ao_out} ;
0176 end
```

[Home](#) > [m](#)


```
0058 end
0059 OUTPUTS = find(Params, 'OUTPUTROWS');
0060 for i_syst=1:length(systs) % for every system
0061     syst = systs(i_syst);
```

```
0108             m.name = [NAME, '_', INPUTNAMES{i}];
0109             m.gain = Gain;
0110             iirs(i_outputs,i_inputs,i_syst) = m;
0111         end
0112     end
0113 end
0114 end
0115 varargout = {iirs};
0116 end
0117
```



```

0001 function varargout = ltpda\_ss\_ss2pz\(varargin\)
0002 % ltpda_ss_ss2pz converts a time continuousvstate space system into a
0003 % pzmodel
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIATION: ltpda_ss_ss2pz converts (an array of) time continuousvstate
space system into a
0007 % pzmodelt. The output field is an array of filter of 3 dimensions. First is
0008 % the output (in the order specified), second the input, third the system.
0009 %
0010 % CALL: pzm = ltpda_ss_ss2pz(systs, Params)
0011 %
0012 % INPUTS: systs - state space subsystem array
0013 % Params - a plist containing the corresponding names of the inputs,
0014 % row number, and output line number.
m CA obnr% the output %%%n7 T(m'r, and output line number.

```

```

0057 if not(length(INPUTS)==length(INPUTCOLUMNS))
0058     msg = 'error because different number of inputs and input columns
specified';
0059     error(msg);
0060 end
0061 OUTPUTS = find(Params, 'OUTPUTROWS');
0062 for i_syst=1:length(systs) % for every system
0063     syst = systs(i_syst);
0064     AMAT = find(syst, 'AMAT');
0065     CMAT = find(syst, 'CMAT');
0066     BMATS = find(syst, 'BMATS');
0067     DMATS = find(syst, 'DMATS');
0068     Ninputs = find(syst, 'NBINPUTS');
0069     NAME = find(syst, 'NAME');
0070     INPUTSIZES = find(syst, 'INPUTSIZES');
0071     INPUTNAMES = find(syst, 'INPUTNAMES');
0072     STEPSIZE = find(syst, 'TIMESTEP');
0073     for i_inputs=1:length(INPUTS) % for every input
0074         j_inputs = 0;
0075         for i=1:Ninputs %finding corresponding indice in the system
0076             if isequal(INPUTNAMES{i}, INPUTS{i_inputs})
0077                 j_inputs = i;
0078             end
0079         end
0080         if 806 re input
0018             error=['error: Iinputname ', INPUTS{i_inputs} , 'f no fouand is
syste '}, TNAM]})
083             lse if(INPUTCOLUMNS({i_inputs})> INPUTSIZE(ji_inputs))Tj T* (0848

```



```
0010 %
0011 % OUTPUTS: 1/0
0012 %
0013 % &&VERSION: $Id: utp_ltpda_loglike.m,v 1.1 2008/03/11 16:52:56 adrien Exp $
0014 %
0015 % HISTORY: 07-03-2008 A Grynagier
0016 %
0017 % TO DO:
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019 ALGONAME = mfilename;
0020 VERSION = '$Id: utp_ltpda_loglike.m,v 1.1 2008/03/11 16:52:56 adrien Exp $';
0021 CATEGORY = 'STATESPACE';
0022 display(['starting ' ALGONAME]);
0023
0024 if not isempty(varargin))
```

0065 end


```
0057     A1 = rand(ss1,ss1);
0058     B1 = rand(ss1,in1);
0059     C1 = rand(out1,ss1);
0060     D1 = rand(out1,in1);
0061     A2 = rand(ss2,ss2);
0062     B2 = rand(ss2,in2);
0063     C2 = rand(out2,ss2);
0064     D2 = rand(out2,in2);
0065     A3 = rand(ss3,ss3);
0066     B3 = rand(ss3,out1);
0067     C3 = rand(out3,ss3);
0068     D3 = rand(out3,out1);
0069     E3 = rand(ss3,out2);
0070     F3 = rand(out3,out2);
0071     o3 = eye(out3);
0072     A_meta = {A1,[],[];[],A2,[];[],[],A3};
0073     B_meta = {[],[],[];[],[],[];B3,E3,[]};
0074     C_meta = {C1,[],[];[],C2,[];[],[],C3};
0075     D_meta = {[],[],[];[],[],[];D3,F3,[]};
0076     Bext_meta = {B1,[],[];B2,[],[]};
```



```
0005 % DESCRIPTION: test function for ltpda_ss_assemble
0006 %
0007 % CALL: utp_ltpda_ss_assemble_2
0008 %
0009 % INPUTS:
0010 %
0011 % OUTPUTS: 1/0
0012 % ***** THERE ARE NO DEFAULT PARAMETERS *****
0013 %
0014 % VERSION: $Id: utp_ltpda_ss_assemble_2.m,v 1.4 2008/03/11 16:52:56 adrien Exp $
0015 %
0016 % HISTORY: 08-02-2008 A Grynagier
0017 % 24-01-2008 A Grynagier
0018 % Creation 16-01-2008 A Grynagier
0019 %
0020 % to do : test with parametric input, test precision on D inversion in
0021 % ss_assemble_fusion
0022 %%%%%%%%%%%%%%%
0023
0024 ALGONAME = mfilename;
0025 VERSION = '$Id: utp_ltpda_ss_assemble_2.m,v 1.4 2008/03/11 16:52:56 adrien Exp $';
```


[Home](#) > [m](#) > [timetools](#) > _____


```

0061     NewTimeStep = plist('TIMESTEP',0.5*Tmax);
0062     j = j+1;
0063     syst3 = ltpda\_ss\_modify\_time\_setup(syst2,NewTimeStep);
0064
0065     NewTimeStep = plist('TIMESTEP',0.05*Tmax);
0066     j = j+1;
0067     syst4 = ltpda\_ss\_modify\_time\_setup(syst3,NewTimeStep);
0068
0069     NewTimeStep = plist('TIMESTEP',0);
0070     j = j+1;
0071     syst5 = ltpda\_ss\_modify\_time\_setup(syst4,NewTimeStep);
0072
0073     NewTimeStep = [NewTimeStep NewTimeStep NewTimeStep];
0074     j = j+1;
0075     syst6 = ltpda\_ss\_modify\_time\_setup([syst5 syst5],NewTimeStep);
0076     display('ok')
0077 catch
0078     display(['error at step ' num2str(j)])
0079     output = 0;
0080     display('pb')
0081 end
0082
0083 %% checking syst2 and sys4 are equal, also for syst0 and syst5
0084 AMAT2 = find(syst2,'AMAT');
0085 AMAT4 = find(syst4,'AMAT');
0086 AMAT0 = find(syst0,'AMAT');
0087 AMAT5 = find(syst5,'AMAT');
0088 % BMATS2 = find(syst2,'BMATS');
0089 % BMATS4 = find(syst4,'BMATS');
0090 % BMATS0 = find(syst0,'BMATS');
0091 % BMATS5 = find(syst5,'BMATS');
0092 % CMAT2 = find(syst2,'CMAT');
0093 % CMAT4 = find(syst4,'CMAT');
0094 % CMAT0 = find(syst0,'CMAT');
0095 % CMAT5 = find(syst5,'CMAT');
0096 if not( norm(AMAT2{1}-AMAT4{1})<5e-7)
0097     display('error possible when changing time between discrete and
discrete')
0098     output = 0;
0099 end
0100 if not(norm(AMAT0{1}-AMAT5{1})<5e-7)
0101     display('error possible when changing time between continuous and
discrete')
0102     output = 0;
0103 end
0104 if not(norm(AMAT2{1}-AMAT4{1})<1e-13)
0105     display(['small error when changing time between discrete and
discrete : ' num2str(norm(AMAT2{1}-AMAT4{1}))])
0106 end
0107 if not(norm(AMAT0{1}-AMAT5{1})<1e-13)
0108     display(['small error when changing time between continuous and
discrete : ' num2str(norm(AMAT0{1}-AMAT5{1}))])
0109 end
0110 end
0111 varargout = {output};
0112 end

```



```

0006 %
0007 % CALL: utp_ltpda_ss_pz2ss
0008 %
0009 % INPUTS: none
0010 %
0011 % OUTPUTS: 1/0
0012 %
0013 % ***** THERE ARE NO DEFAULT PARAMETERS *****
0014 %
0015 % VERSION: $Id: utp_ltpda_ss_pz2ss.m,v 1.4 2008/03/11 16:52:56 adrien Exp $
0016 %
0017 % HISTORY: 22-01-2008 0 Grynagier
0018 %
0019 % TO DO : syntactic (and more numerical tests), consider the jordan bloc/partial
0020 % fraction expansion case for higher precision, or possibly the use of the
0021 % vpa function.
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024 VERSION = '$Id: utp_ltpda_ss_pz2ss.m,v 1.4 2008/03/11 16:52:56 adrien Exp $';
0025 ALGONAME = mfilename;
0026 CATEGORY = 'STATESPACE';
0027 display(['/PACrtiang 'ALGONAME ])
00278 Tj T* (00279 ifnont(isempt([vararina]))Tj T* (00230      ifnisequal( vararina{1},

```


[Home](#) > [m](#) > [timetools](#) > [statespacefunctions](#) > utp_ltpda_ss_random_generate.m

utp_ltpda_ss_random_generate

PURPOSE

utp_ltpda_ss_random_generate subfunction generates a random system

SYNOPSIS

function varargout = utp_ltpda_ss_random_generate(varargin)

DESCRIPTION

utp_ltpda_ss_random_generate subfunction generates a random system
%%%

DESCRIPTION: utp_ltpda_ss_random_generate subfunction generates randomly
a p-th system out of nSysTot systems, with nUTot exogenous inputs,
nSysTot possible 'inner' inputs, OutputSizes and InputSizes neing the
respective sizes of the output sizes of the pSys systems and sizes of the
nUTot input streams.

CALL: utp_ltpda_ss_random_generate

INPUTS: none

OUTPUTS: 1/0


```
0007 %  
0008 % CALL:  ltpda_ss_select()  
0009 %  
0010 % INPUTS:  
0011 %  
0012 % OUTPUTS: 1/0  
0013 %
```

```

0064         for j = 1:inSize(i)
0065             invaranmes_loc{j} = ['INPUT_', num2str(i), '_', num2str(j)];
0066         end
0067         inpuonames{i} = ['INPUT_', num2str(i)];
0068         invarnames{i} = invaranmes_loc;
0069     end
0070     outputvarnames = cell(1, nout);
0071     for i=1:nout
0072         outputvarnames{i} = ['OUTPUT_', num2str(i)];
0073     end
0074     ssvarnames = cell(1, nss);
0075     for i=1:nss
0076         ssvarnames{i} = ['X_', num2str(i)];
0077     end
0078 %% random selection
0079     Outputs = [];
0080     j = 0;
0081     for k=1:nout
0082         if rand()>0.5
0083             j = j+1;
0084             Outputs(j) = k; %#ok<AGROW>
0085         end
0086     end
0087     Inputs = cell(1,nin);
0088     for i=1:nin
0089         j = 0;
0090         inselect = [];
0091         for k=1:inSize(i)
0092             if rand()>0.5
0093                 j = j+1;
0094                 inselect(j) = k; %#ok<AGROW>
0095             end
0096         end
0097         Inputs{i} = inselect;
0098     end
0099 %% inputs and test
0100     syst = plist('TYPE', 'SSSYSTEM' , 'NAME', 'TEST_select' , 'TIMESTEP',
0101 0 , ...
0102         'PARAMNAMES', {} , 'PARAMVALUES', [] , 'PARAMSIGMAS', [] , ...
0103         'NBINPUTS', nin , 'INPUTNAMES', inpuonames , 'INPUTSIZES',
inSize , 'INPUTISUSED', inpuoisused , ...
0104         'INPUTVARNAMES', invarnames , 'OUTPUTVARNAMES', outputvarnames,
'SSVARNAMES', ssvarnames, ...
0105         'AMAT', AMAT , 'BMATS', BMATS , 'CMAT', CMAT , 'DMATS', DMATS );
0106     Ti9 'NBINPUTS', w/7451 /Link ]nTS', DMATS );

```



```
0005 % DESCRIPTION: tests ltpda_simulate_discrete whichsimulates a discrete
0006 % state space system
0007 %
0008 % CALL: utp_ltpda_simulate_discrete
0009 %
0010 % INPUTS:
0011 %
0012 % OUTPUTS: result - 1/0
0013 %
0014 % ***** THERE ARE NO DEFAULT PARAMETERS *****
0015 %
0016 % &&VERSION: $Id: utp_ltpda_ss_simulate.m,v 1.1 2008/03/11 16:52:56 adrien Exp $
0017 %
0018 % HISTORY: 17-02-2008 A Grynagier
0019 %
0020 % TO DO: Numerical and syntactic tests
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 ALGONAME = mfilename;
0024 VERSION = '$Id: utp_ltpda_ss_simulate.m,v 1.1 2008/03/11 16:52:56 adrien Exp $';
```

```
( 'U',ao1, 'N', plist('SIGMA', sigma*8, 'MEAN',mean))];  
0061     ao_out = ltpda\_ss\_simulate(ao_in_list, syst);  
0062  
0063     data_plist = plist('Qs',{0},'Rs',{sigma^2+mean^2},'Noise_Names',{'N'} );  
0064  
0065     ao_in_list2 = plist('U',ao1);  
0066     ao_id = ltpda\_ss\_kalman\_filter(ao_out(1), ao_in_list2(1), data_plist(1),  
syst(1));  
0067  
0068     figure(1)  
0069     plot(ao_out(1))  
0070     figure(2)  
0071     plot(ao_id(1))  
0072 catch  
0073     result = 0;  
0074 end  
0075 varargout = {result};  
0076 end
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

0002 % make is the generic function to decalre or load a subsystem.
0003 %%%
0004 %
0005 % DESCRIPTION: tests ltpda_ss_tomiir
0006 %
0007 % CALL: utp_ltpda_ss_tomiir
0008 %
0009 % INPUTS:
0010 %
0011 % OUTPUTS: 1/0
0012 %


```
0112 display(result)
0113 varargout = {result};
0114 end
0115
```

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

Matlab Index

Matlab Directories

o [m/timetools/statespacefunctions](#)

Matlab Files found in these Directories

Contents	filter	ltpda_pwelch	save
Contents	filtfilt	ltpda_rand	save
save			

[Contents](#)

Contents	get_xy_values	ltpda_zscale	single_operation
Contents	get_xy_values	ltpdagui	slblocks
Contents	get_xyz_values	ltpdalauncher	smoother
Contents	getappdata	ltpdalib	spectrogram
Contents	getlowerFreq	ltpdaquicklook	specwin
Contents	getupperFreq	ltpdasim	specwin_viewer
Contents	gt	md5	split
Contents	hist	mean	sqlResultsGUI
Contents	hist2dot	median	sqrt
Contents	hist2m	method_fcn_template	stairs
Contents	history	mfiltload	std
VisibleProperty	imag	mfir	string
abccascade	index	miir	string
abs	interp	minus	string
acos	inv	minus	string
ancestor	iplot	mkbandpass	string
ao	isappdata	mkbandpass	string
ao2m	isfield	mkbandreject	string
ao_fcn_template	isfield	mkbandreject	string
aoperate	isfield	mkhighpass	string
append	isfield	mkhighpass	string
atan	isfield	mklowpass	string
attachm	isfield	mpower	string
attachmdl	isfield	mrdivide	string
buildConditions	isfield	mtimes	strpad

buildLibrary	isfield	muitable	subsasgn
buildquery	isfield	mux	subsasgn
cat	isfield	mux	subsasgn
cat	isfield	mysql_connect	subsasgn
cat	isfield	mysql_getMD5hash	subsasgn
cat	isfield	mysql_getMaxId	subsasgn
cb_classMethodList	isfield	mysql_getObjIds	subsasgn
cb_condPlusBtn	isfield	mysql_getObjType	subsasgn

display	ltpda_insert	resp	win_HFT196D
display	ltpda_io_collect	resp	win_HFT223D
display	ltpda_isobject	resp	win_HFT248D
display2	ltpda_isuserobject	resp	win_HFT70
downsampl.	ltpda_jacobian	retriev.	win_HFT90D

[export](#)

[ltpda_num2str](#)

Generated on Tue 25-Mar-2008 22:59:47 by [m2html](#) © 2003

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.





[Master index](#)

[Index for classes/@pole](#)



Index for classes/@pole

Matlab files in this directory:



[Contents](#)

%% class: pole %%%%%%%%%%%%%%%%%%%%%%%%%
%%



[cat](#)

CAT concatenate poles into a vector.



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.5723 32.892 sstring.

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.

[oldfs on arraye plot of](matlab:help ao/plot_oldf sg mrdivide)

oldfs on arraye plot of

analysis objects.



Index for classes/@pzmodel

Matlab files in this directory:

[Contents](#)

%% class: pzmodel %%%%%%%%%%%%%%%%%%%%%%%%%
%%

[char](#)

CHAR convert a pzmodel object into a string.

[_____](#)



[< Master index](#)[Index for classes/@cdata >](#)

Index for classes/@cdata

Matlab files in this directory:

[Contents](#)

%% class: cdata %%%%%%%%%
%%%%%%%%%

[cdata](#)

CDATA constant data object class constructor.

[char](#)

CHAR convert a cdata object into a string.

[display](#)

DISPLAY implement terminal display for cdata object.

[_____](#)



Index for classes/@mfir

Matlab files in this directory:

- ☐ [Contents](#) %%%%%%%%%%%%%%% class: mfir %%%%%%%%%%%%%%%
%%%
- ☐ [char](#) CHAR convert a mfir object into a string.
- [display](#) DISPLAY overloads display functionality for mfir objects.
- [eq](#) EQ overloads the == operator for mfir objects.
- [get](#) GET get mfir properties.
- [isfield](#) ISFIELD tests if the given field is one of the object properties.
- ☐ [isvalid](#) ISVALID tests if the given mfir has all the correct fields of the correct type.
- [mfir](#)

Index for classes/@provenance

Matlab files in this directory:

Contents	%% class: provenance %%% %%
_	

Matlab files in this directory:

7

1

10

--	--	--

1





[< Master index](#)

[Index for m/mysql >](#)

Index for m/mysql

Matlab files in this directory:

Contents	%%%%%%%%%%%%%% path: m/mysql %%%%%%%%%%
logindlg	%%%%%%%%%%%%%% LOGINDLG presents the user with a username/password login dialog box.
ltpda_dbquery	

[Home](#) > [m](#) > [mysql](#) of mysql_conn.m (m)Tj ET EMCH1Div <</MC6D 5 >>BDC rg 1TT0 1 20.0782956 020.0782988 8.1


```
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: MYSQL_CONNECT connects to an AO repository and returns the
0007 %               connection object.
0008 %
0009 % CALL:         conn = mysql_connect(hostname)  % connects to 'test' database
```


Index for classes/@plist

Matlab files in this directory:

[Contents](#) %%%%%%%%%%%%%%% class: plist %%%%%%%%%%%%%%%
%%%%%%%%

ð [private](#)

Generated on Tue 25-Mar-2008 22:59:47 by

[!\[\]\(d0a1791f26d167e866e44ebbf83efebe_img.jpg\) Master index](#)

[Index for classes/@pole!\[\]\(5eb1325dfdc3f1cad8426726c0db51cd_img.jpg\)](#)

Index for classes/@pole

Matlab files in this directory:

[!\[\]\(950a62bbddad88d64435fd35607dfc42_img.jpg\) Contents](#) %%%%%%%%%%%%%%% class: pole %%%%%%%%%%%%%%%





 [export](#)

EXPORT export an analysis object to a text file.





[Home](#)

of the target transfer function

```
0008 %
0009 % A Monsky 24-07-07
0010 %
0011 % $Id: ngconv.m,v 1.2 2008/01/23 17:23:56 anneke Exp $
0012
0013
0014
0015 zs = get(pzm, 'zeros');
0016 ps = get(pzm, 'poles');
0017
0018 %%
0019 f_zer = [];
0020 q_zer = [];
0021 for j=1:length(zs)
0022     z = zs(j);
0023     f_zer(j,1) = [get(z, 'f')];
0024     q_zer(j,1) = [get(z, 'q')];
0025     %if isnan(q_zer(j))
0026     %     q_zer(j,1) = 0;
0027     %end
0028     %zv(j,1:2) = [f q];
0029 end
0030
0031 f_pol = [];
0032 q_pol = [];
0033 for j=1:length(ps)
0034     p = ps(j);
0035     f_pol(j,1) = [get(p, 'f')];
0036     q_pol(j,1) = [get(p, 'q')];
0037     %if isnan(q_pol(j))
0038     %     q_pol(j,1) = 0;
0039     %end
0040 end
0041 %% calculate factors from f and q
0042 pol = fq2fac(f_pol,q_pol);
0043 zer = fq2fac(f_zer,q_zer);
0044 %%
0045 [b,a] = conv_noisegen(pol,zer);
0046
0047
0048
0049 %END
```



```
0010 n = length(Tinit);  
0011 %% writing the generator  
0012 r = randn(n,1);  
0013 y = Tinit * r;
```

Generated on Tue 25-Mar-2008 23:00:00 by [_____](#)

[Home](#) > [m](#) > [noisegenerator](#) > ngprop.m

ngprop

PURPOSE [^]

0004 H ngpropcouli]/e used as EnP:ET /T*ng1ro8

[Home](#) > > [sigproc](#) > [frequency_domain](#)


```
0054 % fix vectors
0055 x = x(:);
0056 win = win(:);
0057
0058 % Initialise the spectral estimates
0059 Sxx = zeros(nfft,1);
0060
0061 % loop over segments
0062 LminusOverlap = L-noverlap;
0063 xStart = 1:LminusOverlap:k*LminusOverlap;
0064 xEnd = xStart+L-1;
0065
0066 for i = 1:k,
0067     % data segment, detrend and window
0068     if order < 0
0069     else
0070         [y,coeffs] = ltpda_polyreg(x(xStart(i):xEnd(i)), order);
0071         dseg = y.*win;
0072     end
0073     % compute fft
0074     xx = fft(dseg,nfft);
0075     Sxx = ((Sxx .* (i-1)) + xx.*conj(xx)) ./ i;
0076 end
0077 % take only one-sided
0078 Sxx = Sxx(1:nfft/2+1);
0079
```


 [tpda_memo](#)

/Link <</MCID 2>>BDC 001RG6per72 W q 1 0 0109.648 32.5 791.4549 c
precision at a central place.


```
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 %
```

Index for m/timetools/statespacefunctions

Matlab files in this directory:



[Contents](#)

%% path: m/
timetools/statespacefunctions %%%%%%%%%
%%%%%%%%%



[ltpda_jacobian](#)

subfunction to handle exceptions when calling the
jacobian function



[ltpda_loglike](#)

Log likelihood criterion



[ltpda_ss_assemble](#)

assembles embedded subsystems, with exogenous
inputs



[< Master index](#)

[Index for m/timetools >](#)

Index for m/timetools

Matlab files in this directory:

[< _____](#)

[Home](#) > [m](#) > [timetools](#) > _____



[Master index](#)

[Index for classes/@fsdata/plivate](#)

Index for classes/@fsdata/plivate

Matlab files in this directory:



[Contents](#)

%% class: fsdata/plivate %%%%%%%%%%%%%%%%%%%%%%%%%
%%



[retrieve](#)

RETRIEVE fsdata objects with given ids from the LTPDA repository specified
by the input

Subsequent directories:

o [CVS](#)

Generated on Tue 25-Mar-2008 22:59:47 by [m2html](#) © 2003

Generated on Tue 25-Mar-2008 22:59:47 by [m2html](#) © 2003

[<](#) Master index

Index for classes/@miir/private [>](#)

Index for classes/@miir/private

[< Master index](#)[Index for classes/@provenance >](#)

Index for classes/@provenance

Matlab files in this directory:

 Contents	%% class: provenance %%%%%%%%%%%%%%%%%%%%%%%%% %%
 char	CHAR convert a provenance object into a string.
 display	DISPLAY overload terminal display for provenance objects.
 eq	EQ overloads the == operator for provenance objects.
 get	

[<](#) Master index

Index for classes/@provenance/private [>](#)

Index for classes/@provenance/private



 [win_Nuttall3](#)





[Master index](#)

[Index for classes/@tsdata/private](#)



Index for classes/@tsdata/private

Matlab files in this directory:



[Contents](#)

%% class: tsdata/private %%%%%%%%%%%%%%%%%%%%%%%%%
%%



[retrieve](#)

RETRIEVE tsdata objects with given ids from the LTPDA repository specified
by the input

Subsequent directories:

0 stories:

[< Master index](#)

[Index for classes/@xydata >](#)

Index for classes/@xydata

Matlab files in this directory:



[< Master index](#)

[Index for classes/@xydata/private >](#)

Matlab files in this directory:

	Contents	%% class: xydata/private %%%%%%%%%%%%%%%%%%%%%%%%% %%
	retrieve	RETRIEVE xydata objects with given ids from the LTPDA repository specified by the input

Subsequent directories:

[< Master index](#)

[Index for classes/@zero/plivate >](#)

Index for classes/@zero/plivate

Matlab files in this directory:

[<](#) Master index

Index for m/etc [>](#)

[<](#) Master index

Index for m/gui/ao_browser [□](#)

[<](#) Master index

Index for m/gui/gltpda

Matlab files in this directory:

 [Contents](#)

 [buildLibrary](#) Function to update the library with all the methods functions.

 [g_armmult](#) This is the automatic wrapper



%%%%%%%%%%%%%% path: m/gui/gltpda %%%%%%%%%%%%%%%
%

[<](#) Master index

 [getWorkspaceObjs](#)

Get a list of LTPDA objects in the MATLAB workspace

[<](#) Master index

[< Master index](#)

[Index for m/gui/quicklook >](#)

Index for m/gui/quicklook

Matlab files in this directory:

 [Contents](#) %%%%%%%%%%%%%%% path: m/gui/quicklook %%%%%%%%%%
%%%%%%%%%

 [ltpdaquicklook](#) LTPDAQICKLOOK allows the user to quicklook LTPDA objects.

Subsequent directories:

- [CVS](#)
- [generic_class_fcn](#)

Generated on Tue 25-Mar-2008 22:59:47 by [m2html](#) © 2003







[<](#) Master index

Directory m/sigproc/frequency_domain

[< Master index](#)

[Index for m/sigproc/time_domain >](#)

Index for m/sigproc/time_domain

Matlab files in this directory:oc/time_doma%%%%%%%%%

Index for m/timetools

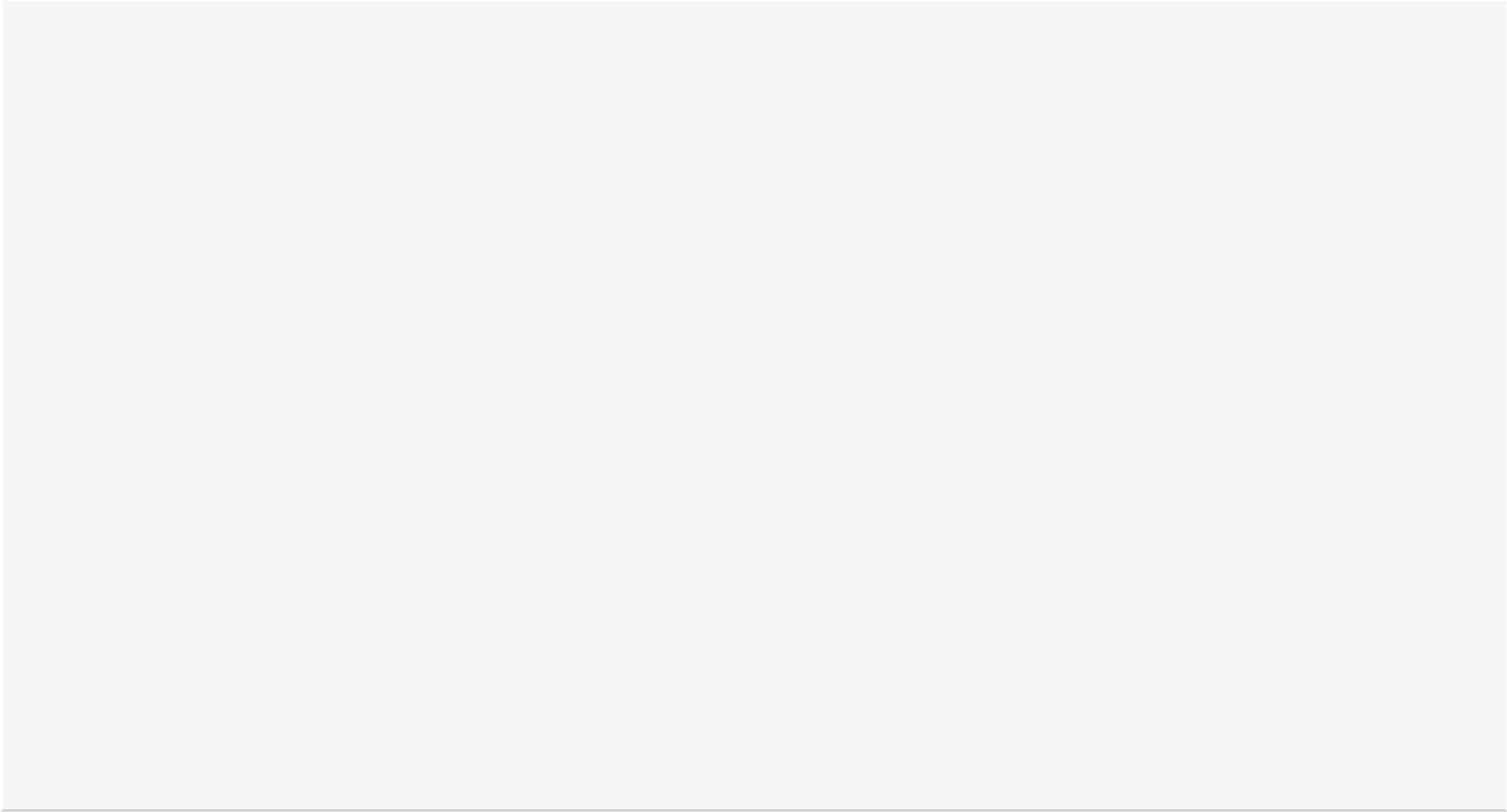
Matlab files in this directorI:

 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:

- o [CVS](#)
- o [statespacefunctions](#)

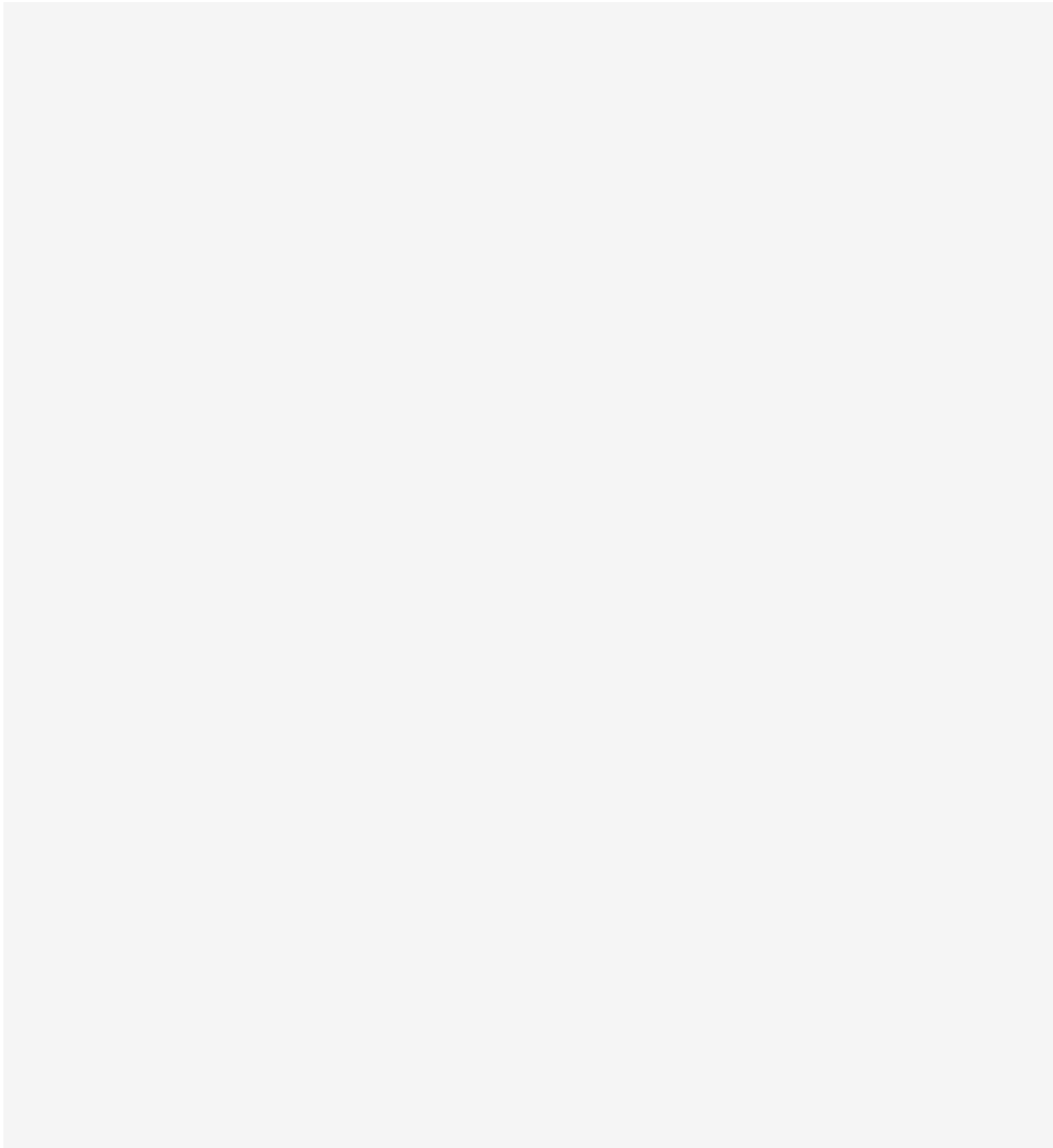
Generated on Tue 25-Mar-2008 22:59:47 by [m2html](#) © 2003



--	--

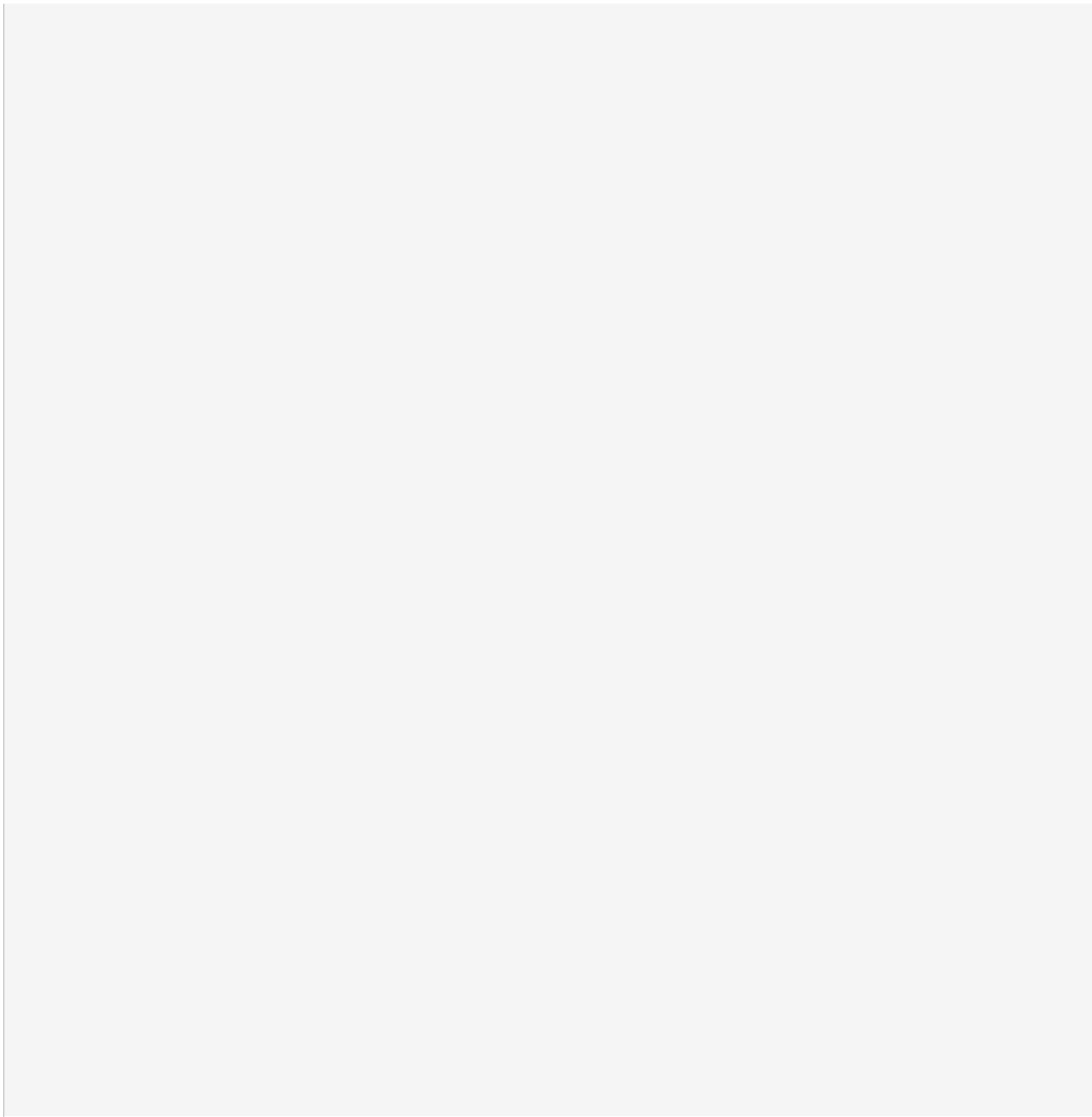
--	--

--

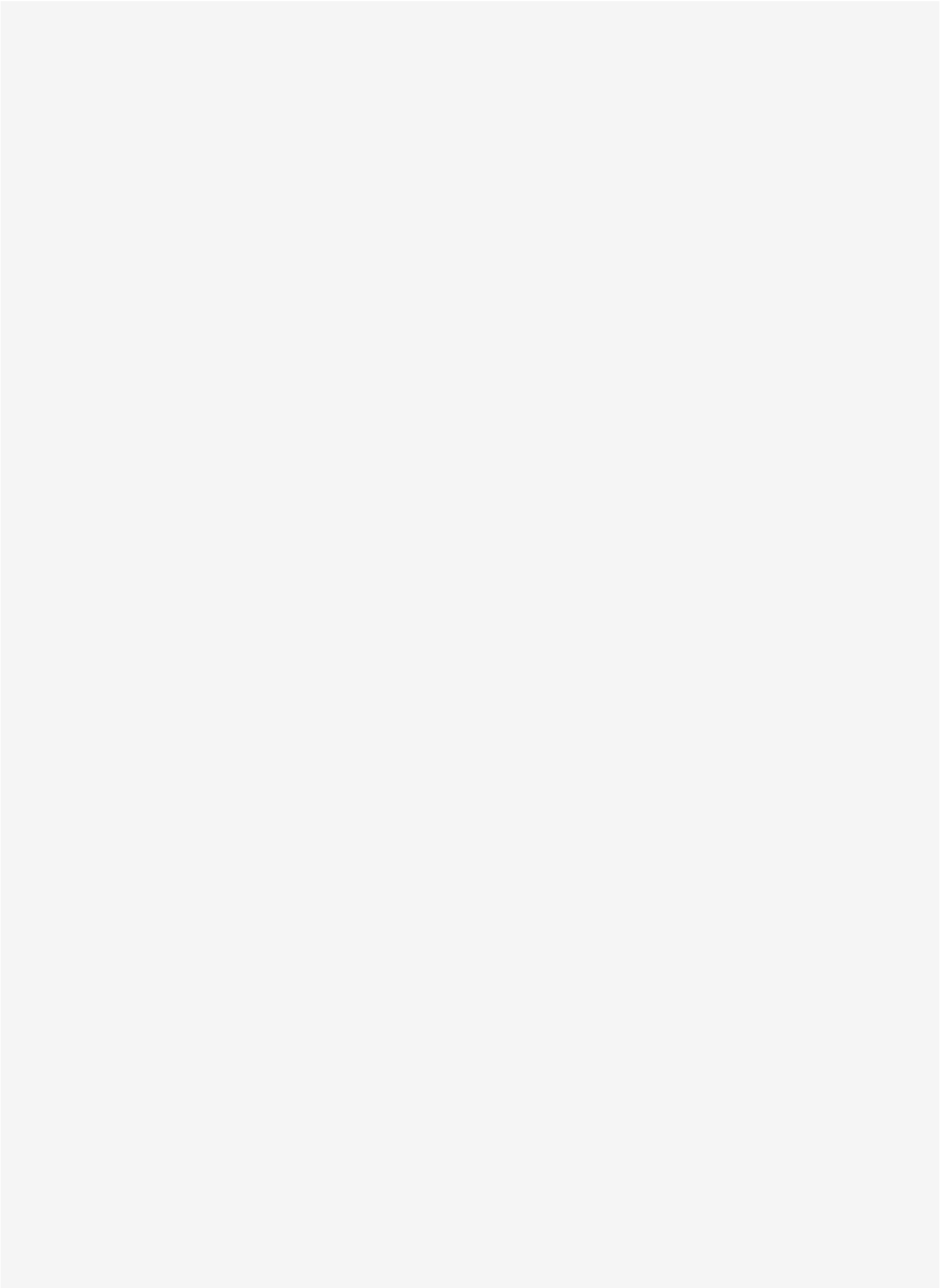


0058

0059 l2=sp(:,1)<=e;



fq2fac

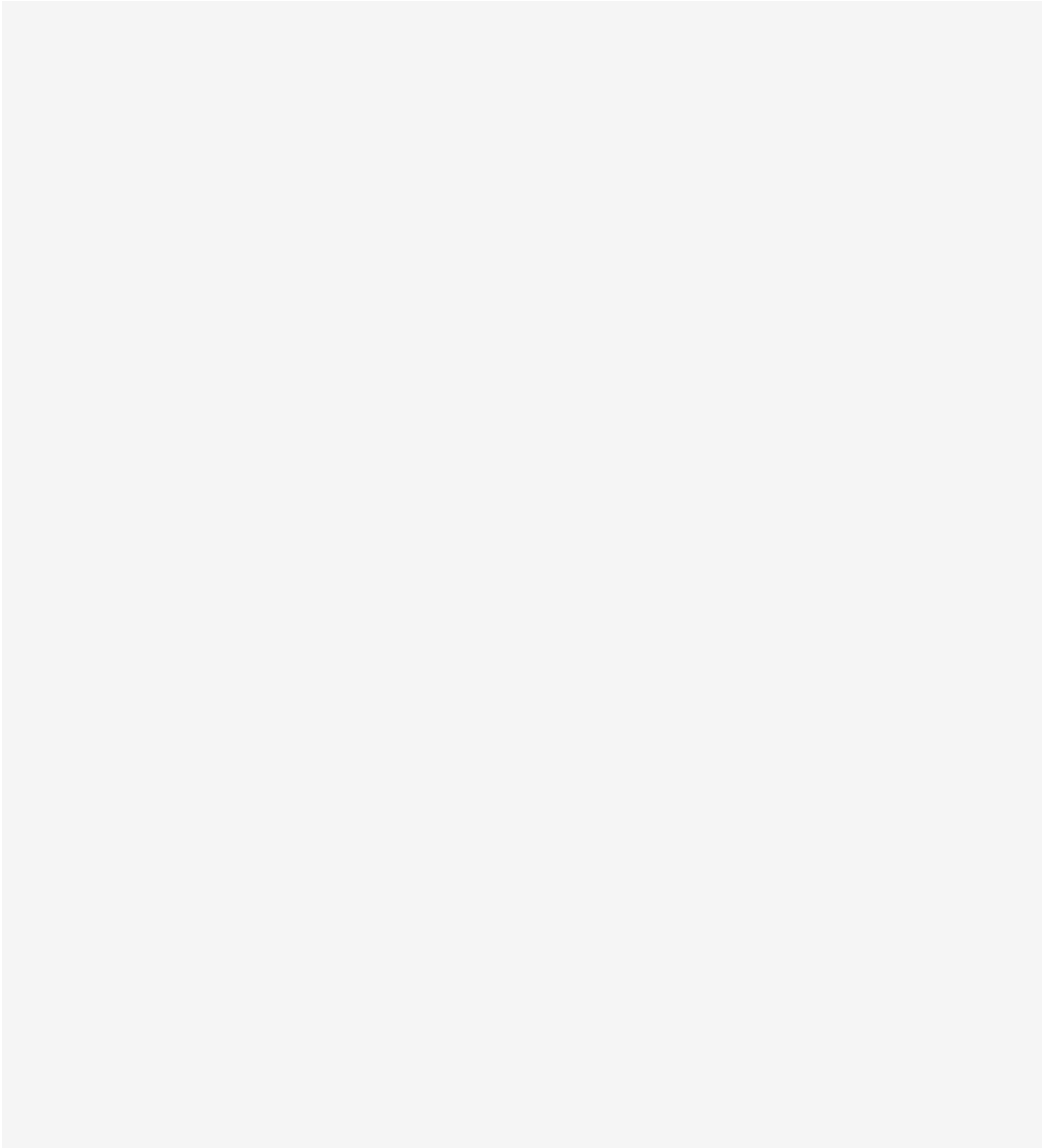


Generated on Tue 25-Mar-2008 23:00:00 by


```
0010 % $Id: g_copyparams.m,v 1.1 2008/03/01 13:43:20 nicola Exp $
0011
0012 setup(block);
0013
0014 %%
0015 function setup(block)
0016
0017     %% Register dialog parameter: none, because they're retrieved directly
0018     %% from the memory. This will prevent the user to modify the parameters
0019     %% outside the proper parameters panel:
0020     block.NumDialogPrms = 0;
0021
0022
```


--	--


```
0068         iplot(LTPDAinvar{block.InputPort(1).Data(1)},LTPDAinvar{block.InputPort
(1).Data(2)});
0069         case0068         iplot(LTPDAinvar{block.InputPort(1).Data(1)},LTPDAinvar{blo
casecasecase
```




```

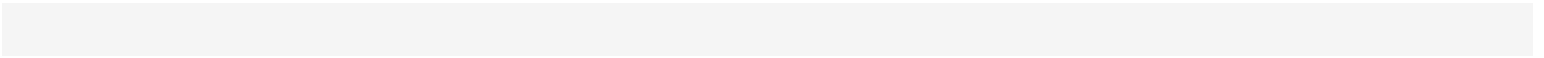
0010 % $Id: g_plot.m,v 1.1 2008/03/01 13:43:20 nicola Exp $
0011
0012 setup(block);
0013
0014 %%
0015 function setup(block)
0016 global AnOb;
0017
0018 %% Register dialog parameter: none, because they're retrieved directly
0019 %% from the memory. This will prevent the user to modify the parameters
0020 %% outside the proper parameters panel:
0021 block.NumDialogPrms = 0;
0022
0023 %% Register number of input and output ports
0024 block.NumInputPorts = 1;
0025 block.NumOutputPorts = 1;
0026
0027 %% Setup functional port properties to dynamically inherited.
0028 block.SetPreCompInpPortInfoToDynamic;
0029 block.SetPreCompOutPortInfoToDynamic;
0030
0031 block.InputPort(1).DirectFeedthrough = true;
0032 block.InputPort(1).DatatypeID = 0;
0033 block.InputPort(1).Complexity = 0;
0034 % block.InputPort(1).Dimensions = 2;
0035 block.OutputPort(1).DatatypeID = 0;
0036 block.OutputPort(1).Complexity = 0;
0037 % block.OutputPort(1).Dimensions = 1;
0038 block.SampleTimes = [0 0];
0039 block.SetAccelRunOnTLC(false);
0040
0041 %% Register methods
0042 block.RegBlockMethod('SetInputPortSamplingMode', @SetInpPortFrameData);
0043 block.RegBlockMethod('SetInputPortDimensions', @SetInpPortDims);
0044 block.RegBlockMethod('SetOutputPortDimensions', @SetOutPortDims);
0045 block.RegBlockMethod('Outputs', @Outputs);
0046
0047 function SetInpPortFrameData(block, idx, fd)
0048 block.InputPort(1).SamplingMode = fd;
0049 block.OutputPort(1).SamplingMode = fd;
0050
0051 function SetInpPortDims(block, idx, di)
0052 block.InputPort(idx).Dimensions = di;
0053
0054 function SetOutPortDims(block, idx, di)
0055 block.OutputPort(idx).Dimensions = di;
0056
0057
0058 %%
0059 function Outputs(block)
0060 global AnOb
0061
0062 y=length(block.InputPort(1).Data);
0063 figure

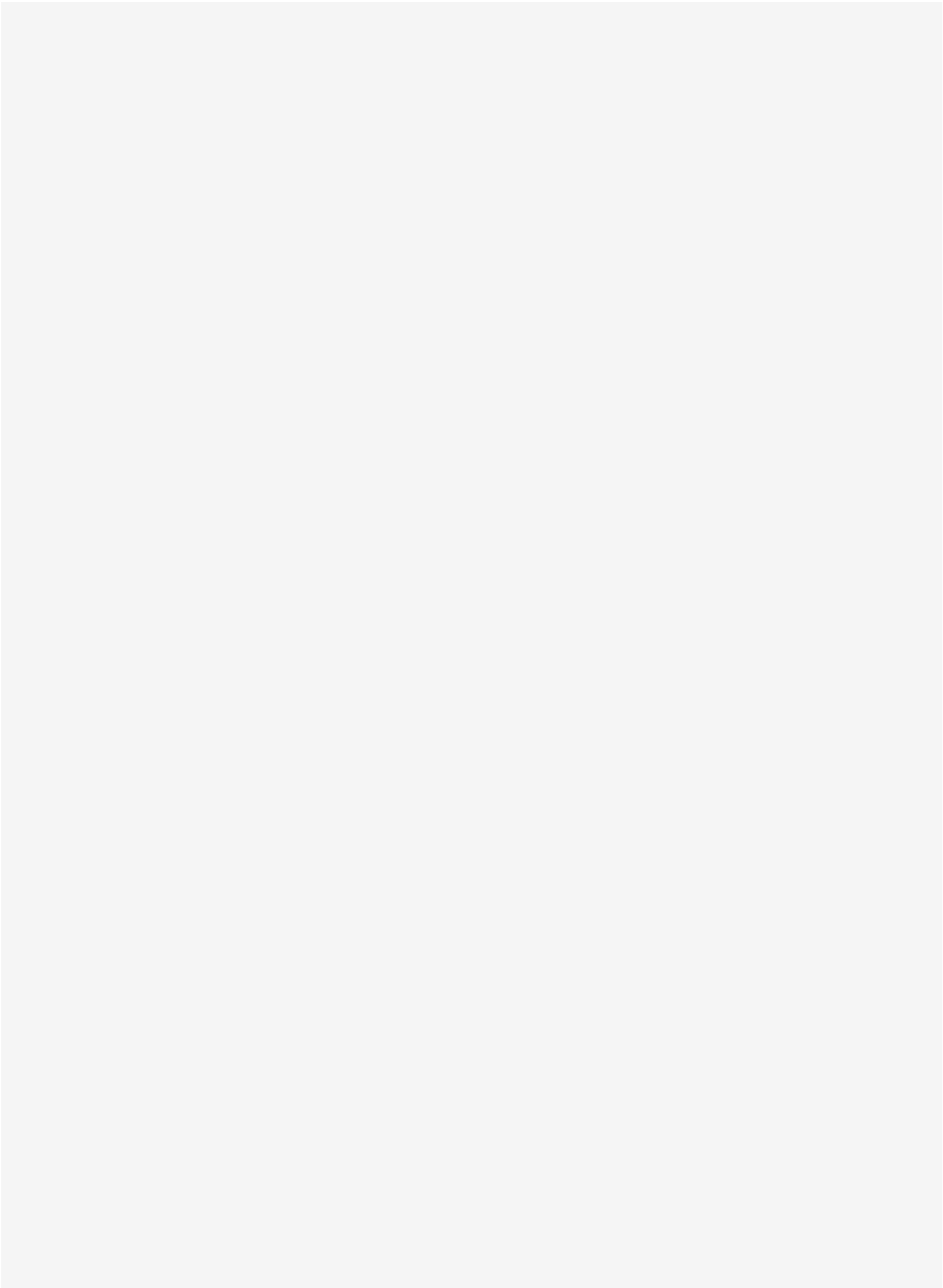
```





```
0069     end
0070
0071     outdata=resp(data,currparam);
0072     outdata = num2cell(outdata);
0073
0074     xx = size(LTPDAinvar,1);
0075     LTPDAinvar(xx+1,:) = [outdata,0];
```





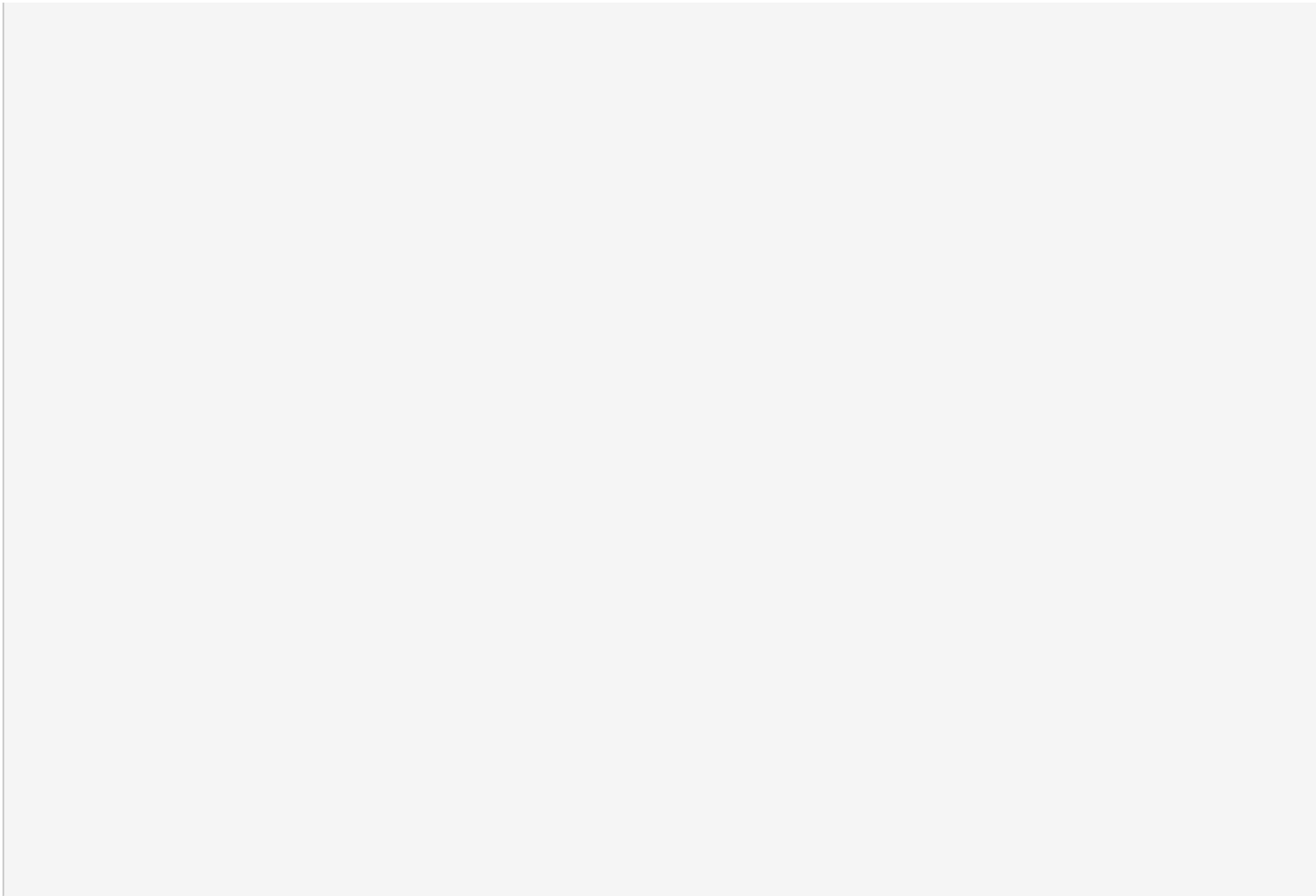
Contents

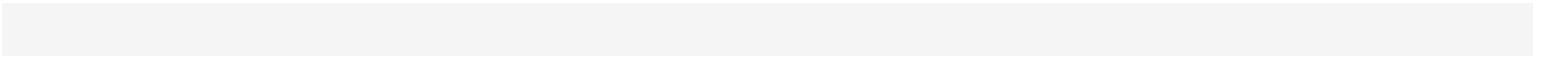
PURPOSE ☐

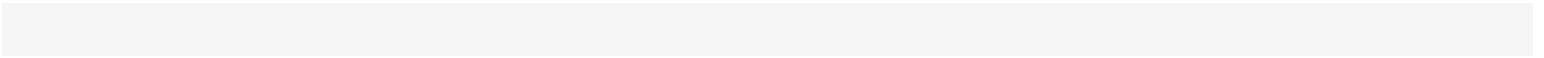
This function is called by:

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003


```
(obj1), numel(obj1), numel(obj2)));  
0057     result = 0;  
0058     return  
0059 end  
0060  
0061 exception_list = varargin;  
0062  
0063 %% Convert a potential existing plist into a exception  
0064 if ~isempty(exception_list) && isa(exception_list{1}, 'plist')  
0065     exception_list = find(exception_list{1}, 'Exceptions');  
0066     if isempty(exception_list)  
0067         exception_list = cell(0);  
0068     elseif
```








```

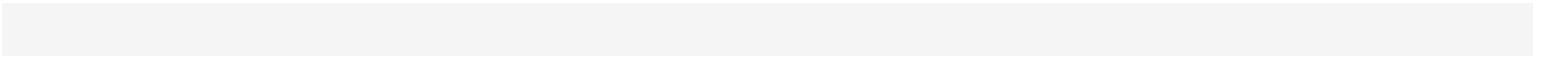
0016
0017 %%% REMARK: The classes
0018 %%%
0019 %%%         ao,
0020 %%%         cdata,
0021 %%%         pole,
0022 %%%         time,
0023 %%%         timeformat,
0024 %%%         timespan,
0025 %%%         zero,
0026 %%%         don't use this generic function.
0027
0028 fields = fieldnames(objs);
0029
0030 %%% Plausibility check: The default parameter list must contain all fieldnames
0031 if length(fields) ~= nparams(DEFAULT_PLIST)
0032     error('\n### The DEFAULT_PLIST doesn't contain all field names of the %-
class!', class(objs));
0033 end
0034
0035 %%% CThe if this is a special call: default parameter list, cvs-version,
category
0036 if length(propArgin) == 1 && ischar(propArgin{1})
0037     in = propArgin;
0038     if strcmp(in, 'Params')
0039         objs = DEFAULT_PLIST;
0040         return
0041     elseif strcmp(in, 'Version')
0042         objs = VERSION;
0043         return
0044     elseif strcmp(in, 'Category')
0045         objs = CATEGORY;
0046         return
0047     end
0048 end
0049
0050 if isempty(propArgin)
0051     propArgin = {plist()};
0052 end
0053
0054 if nargin >= 2 && isa(propArgin{1}, 'plist')
0055     pl = propArgin{1};
0056     propArgin = {};
0057     for ii = 1:nparams(pl)
0058         %%% The 'key' name of a plist/param is always stored with upper case
0059         %%% characters. To set a ao property with this key we have to lower the
0060         %%% 'key' characters.
0061         propArgin{end+1} = lower(pl.params(ii).key);
0062         propArgin{end+1} = pl.params(ii).val;
0063     end
0064 end
0065
0066 while length(propArgin) >= 2
0067     prop = propArgin{1};
0068     val = propArgin{2};
0069     propArgin = propArgin(3:end);
0070
0071 %%% For all objects

```



```
a>      - e0eturn HFT248D window sample for input z.
<a href="matlab:help specwin/private/win_HFT70">specwin/private/win_HFT70</
a>      - e0eturn HFT70 window sample for input z.
<a href="matlab:help specwin/private/win_HFT90D">specwin/private/win_HFT90D</
a>      - e0eturn HFT90D window sample for input z.
<a href="matlab:help specwin/private/win_HFT95">specwin/private/win_HFT95</
a>      - e0eturn HFT95 window sample for input z.
<a href="matlab:help specwin/private/win_Hamming">specwin/private/win_Hamming</
a>      - e0eturn Hamming window sample for input z.
<a href="matlab:help specwin/private/win_Hanning">specwin/private/win_Hanning</
a>      - e0eturn Hanning window sample for input z.
<a href="matlab:help specwin/private/win_Nuttall3">specwin/private/win_Nuttall3</
a>      - e0eturn Nuttall3 window sample for input z.
<a href="matlab:help specwin/private/win_Nuttall3a">specwin/private/
win_Nuttall3a</a>      - e0eturn Nuttall3a window sample for input z.
<a href="matlab:help specwin/private/win_Nuttall3b">specwin/private/
win_Nuttall3b</a>      - e0eturn Nuttall3b window sample for input z.
<a href="matlab:help specwin/private/win_Nuttall4">specwin/private/win_Nuttall4</
a>      - e0eturn Nuttall4 window sample for input z.
<a href="matlab:help specwin/private/win_Nuttall4a">specwin/private/
win_Nuttall4a</a>      - e0eturn Nuttall4a window sample for input z.
<a href="matlab:help specwin/private/win_Nuttall4b">specwin/private/
win_Nuttall4b</a>      - e0eturn Nuttall4b window sample for input z.
<a href="matlab:help specwin/private/win_Nuttall4c">specwin/private/
win_Nuttall4c</a>      - e0eturn Nuttall4c window sample for input z.
<a href="matlab:help specwin/private/win_Rectangular">specwin/private/
win_Rectangular</a> - e0eturn rectangular window sample for input z.
<a href="matlab:help specwin/private/win_SFT3F">specwin/private/win_SFT3F</
a>      - e0eturn SFT3F window sample for input z.
<a href="matlab:help specwin/private/win_SFT3M/private/win_SFT3F7r>47tCm416/4.]5SFT3
5 <a href="matlab:help specwin/private/win_SFT3M/private/win_SFT3F7r>47tCm416/4.]5SF5
```

Generated on Tue 25-Mar-2008 23:00:00 by



```

0022
0023 REPO_GUI_SERVERLIST = {'localhost', '130.75.117.67', '130.75.117.61'}; % A list
of repositories presented
0024                                     % to the user
0025 REPO_GUI_FONTSIZE    = 10;          % Choose Repository GUI font size
0026 PASSWD_WIN_FONTSIZE = 12;          % Choose login/passw dialog window font size
0027
0028 DBDRIVER = 'com.mysql.jdbc.Driver'; % Set LTPDA Repository database driver
0029
0030 WRAP_STRINGS          = 8;          % Wrap strings at this value
0031 WRAP_LEGEND_STRINGS   = 10;         % Wrap legend strings at this value
0032 XML_SET_SIZE          = 50000;      % Set the maximum line length for writing
data sets to XML
0033
0034 TIMEZONE              = 'UTC';      % Users timezone
0035
0036
0037 TIME_FORMAT_STR = 'yyyy-mm-dd HH:MM:SS.FFF'; % Users time format.
0038
0039 DEFAULT_DATA_DIR = 'C:\data';
0040 DEFAULT_ANALYSES_DIR = 'C:\analyses';
0041
0042 %-----
0043 %-----
0044 % NO NEED TO EDIT BELOW HERE
0045 %-----
0046 %-----
0047
0048 v = ver('LTPDA');
0049
0050 logo = {...
0051 '
0052             ****
0053             **
0054             -----
0055             \\\ 0.133 0.' ,...
0056             // 133 0.' ,...
0057             |Tj 0.133 0../ |33 0.' ,...
0058             ** |33 0+----+../ +----+.. |3**' ,...
0059             *** | 33 0|//-----| |33 0| ***' ,...
0060             ** |33 0+----+..33 0../+----+.. |3**' ,...
0061             |Tj 0.133 0.. 0../ |33 0.' ,...
0062             \\ 0.133 0.0../ ///' ,...
0063             \\ 0.133// ///' ,...
0064             -----
0065             **
0066             ****
0067 };
0068
0069 l1 = '+-----+';
0070 ll = length(l1);
0071
0072 disp(l1);
0073 disp('|Tj 0.133 0.. 0.....|')
0074 for j=1:length(logo)
0075     disp([strpad(sprintf('|Tj 0.1%s0.', char(logo{j})), ll-1) '|']);
0076 end
0077 disp([strpad('|', ll-1) '|'])

```

```
0078 disp([strpad(sprintf('|'Welcome to %s', v.Name), ll-1) '|''])
0079 disp([strpad(''|'
```

```
0135 javaaddpath(mysqldriver);  
0136  
0137
```


HISTORY: 26-01-2007 M Hewitson
Creation

%%

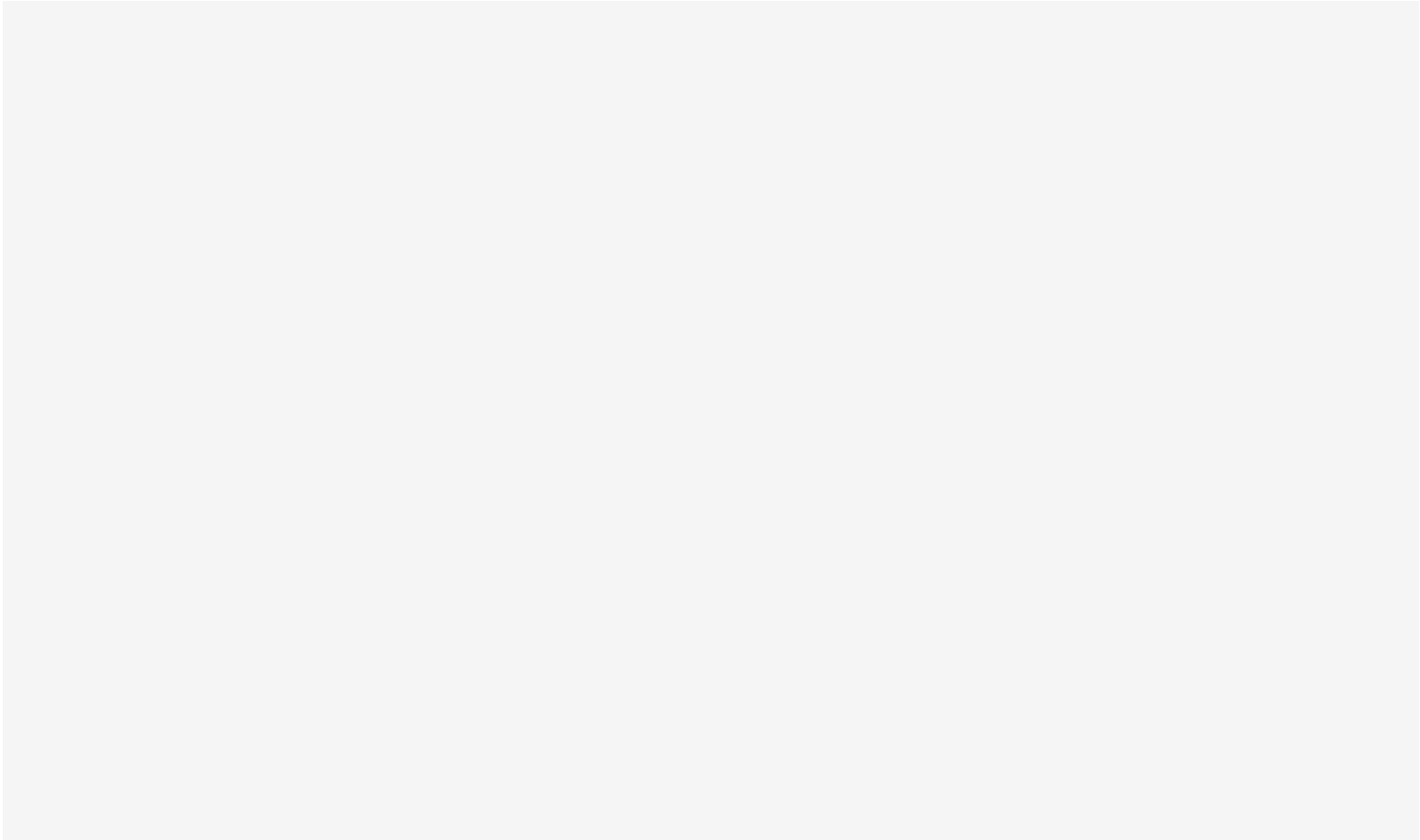
CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

SOURCE CODE

0001



```
%%%%%%%%%%%%%% class: timeformat %%%%%%%%%%%%%%%
```

This is a script file.

```

%%%%%%%%%%%%%% class: timeformat %%%%%%%%%%%%%%%

timeformat/char - convert a
timeformat to a char array.

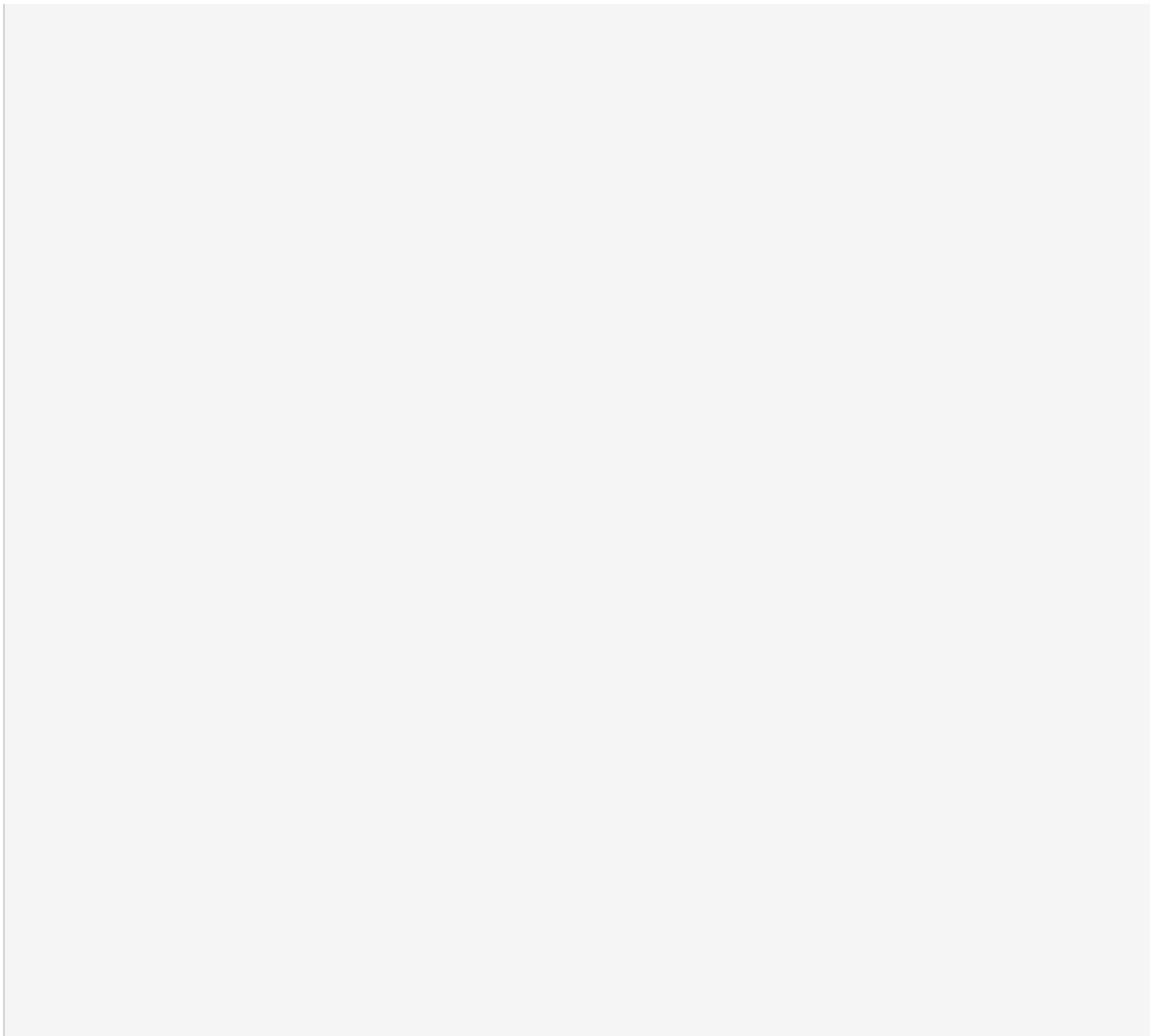
```

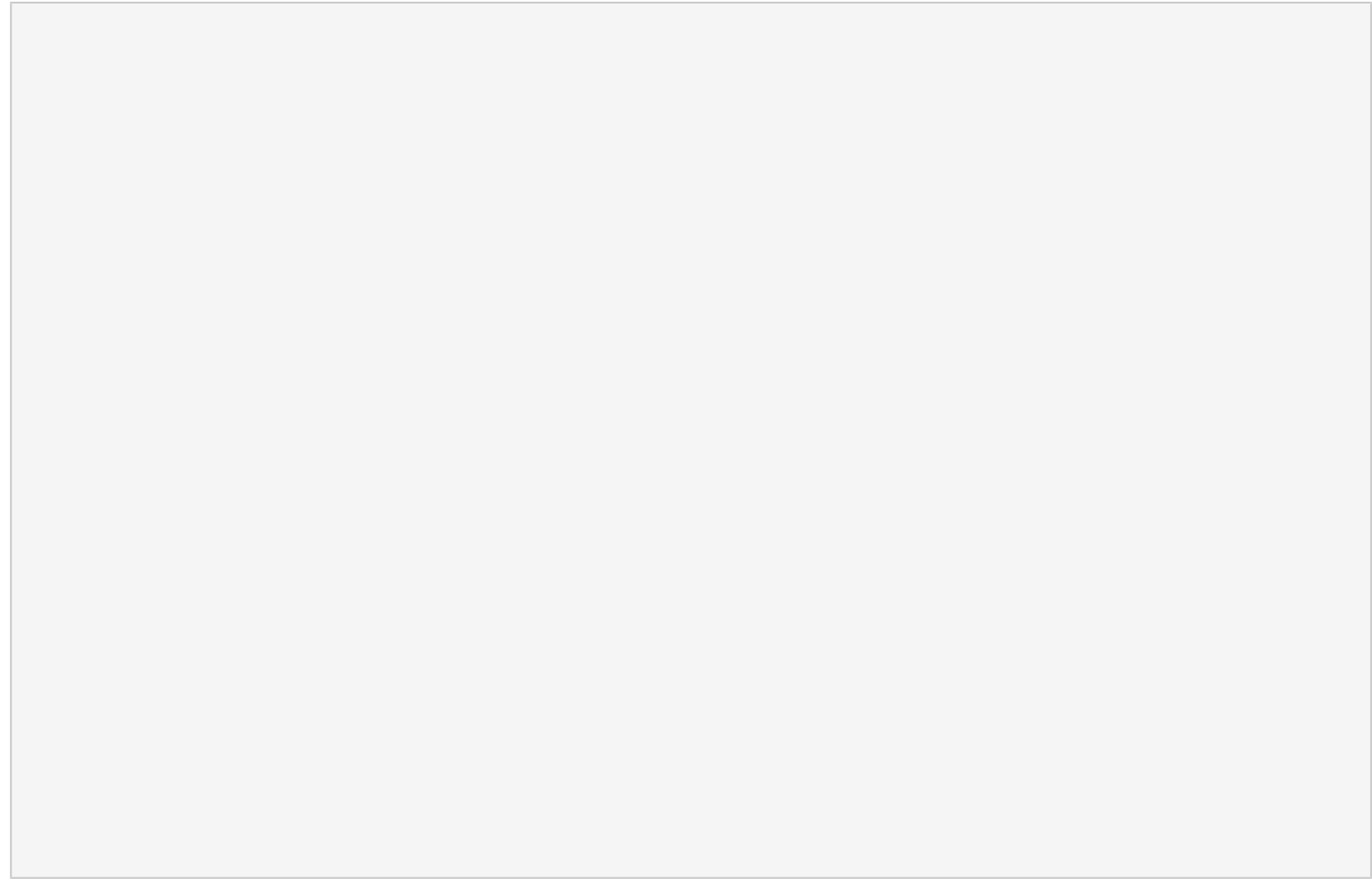
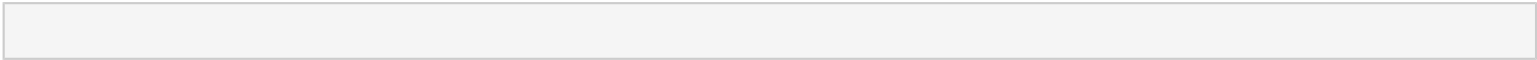


```
0010 % INPUTS:      str1 - string
0011 %              str2 - string
0012 %              ... 0.961 01 1 0ing
```

--	--

--	--





Adapted for LTPDA

\$Id: ltpda_uitree.m,v 1.2 2008/01/22 20:45:22 hewitson Exp \$

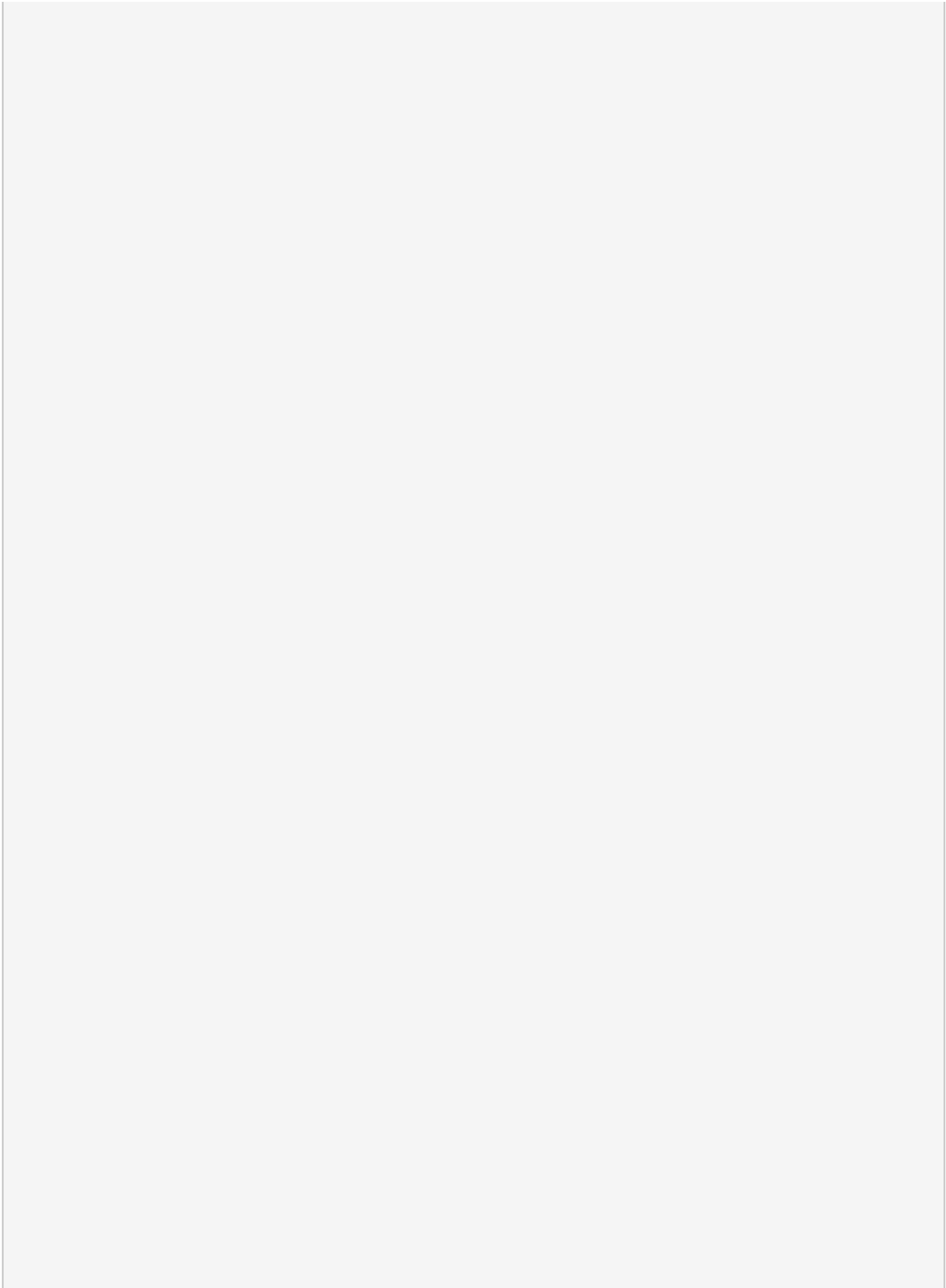

```
0086 %           'define an ExpandFcn for the nodes.']);
0087 %       end
0088 %
0089 %       if e 0.133 rg (%)Tj 0 70 0T== 0)89 %           'define an Expa0 rg =
0089 %       if e 0.133AdDC gin%           'def89 %           'defi$Id:C ET /\
```



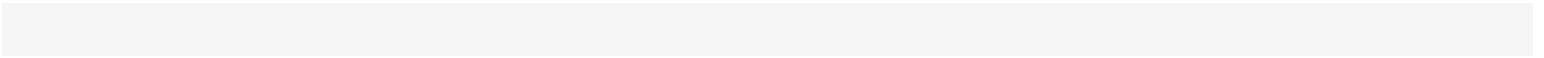
```

0201
0202 evdnode = evd.getCurrentNode;
0203 % indices = [];
0204
0205 if ~tree.isLoaded(evdnode)
0206     value = evdnode.getValue;
0207
0208     % <call a user function(value) which returns uitreenodes>;
0209     cbk = expfcn;
0210     if iscell(cbk)
0211         chldnodes = feval(cbk{1}, tree, value, cbk{2:end});
0212     else
0213         chldnodes = feval(cbk, tree, value);
0214     end
0215
0216     if (length(chldnodes) == 1)
0217         % Then we dont have an array of nodes. Create an array.
0218         chnodes = chldnodes;
0219         chldnodes = javaArray('com.mathworks.hg.peer.UITreeNode', 1);
0220         chldnodes(1) = java(chnodes);
0221     end
0222
0223     tree.add(evdnode, chldnodes);
0224     tree.setLoaded(evdnode, true);
0225 end
0226
0227 end
0228
0229 %% -----
0230 function nodeSelected(src, evd, tree, selfcn)                                %#ok
0231 cbk = selfcn;
0232 hgfeval(cbk, tree, evd);
0233
0234 end
0235
0236 %% -----
0237 function [node, expfcn] = processNode(root)
0238 expfcn = [];
0239
0240 if isempty(root) || isa(root, 'com.mathworks.hg.peer.UITreeNode') || ...
0241     isa(root, 'javahandle.com.mathworks.hg.peer.UITreeNode')
0242     node = root;
0243 elseif ishghandle(root)
0244     % Try to process as an HG object.
0245     try
0246         node = uitreenode(handle(root), get(root, 'Type'), ...
0247             [], isempty(get(0, 'children')));
0248     catch
0249         node = [];
0250     end
0251     expfcn = @hgBrowser;
0252 elseif ismodel(root)
0253     % Try to process as an open Simulink system
0254
0255     % TODO if there is an open simulink system and a directory on the path with
0256     % the same name, the system will hide the g rg eerhaps Theshoulg (end)Tj 0 0

```



0315



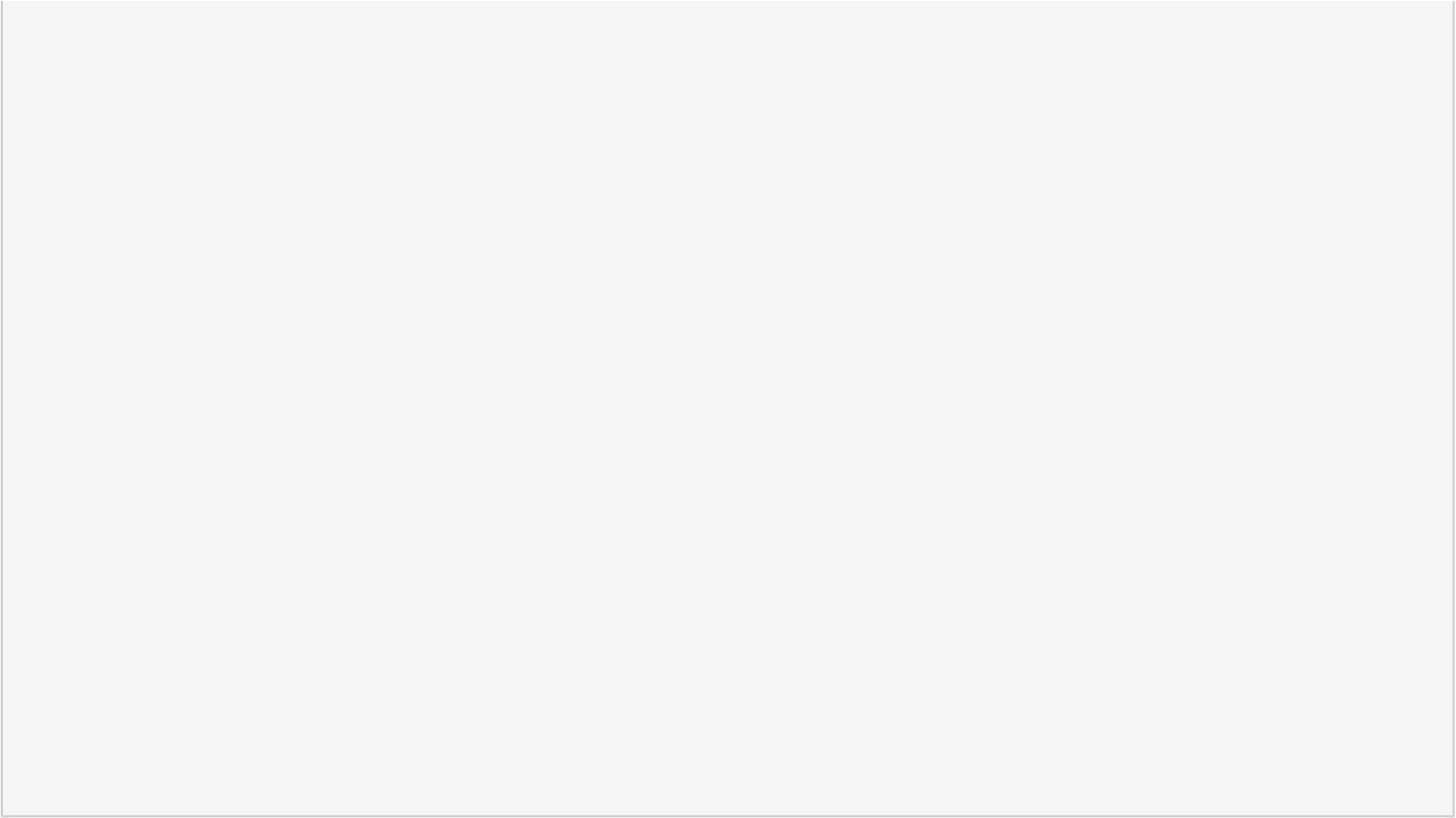
--

This function is called by:

- `jcontrol` JCONTROL constructor for JCONTROL class
- `set` SET method overloaded for JCONTROL class

SOURCE CODE

```
0051     {Tproperty/value pairs in structure
0052     {TGet the names
0053     propnames=fieldnames(varargin{2});
0054     {Tget the values
0055     propvalues=cell(length(propnames),1);
0056     for i=1:length(propnames)
0057         propvalues{i}=varargin{2}.(propnames{i});
0058     end
0059     {TRecursive call to set
0060     set(varargin{1}, propnames, propvalues);
0061     return
0062 elseif nargin==3 && iscell(varargin{2}) && iscell(varargin{3})
0063
```



--


```
0009 % CALL:          classes = ltpda_user_classes()
0010 %
0011 % INPUTS:
0012 %
0013 % OUTPUTS:        classes - a cell array with a list of recognised LTPDA object
types
0014 %
0015 % VERSION:         $Id: ltpda_user_classes.m,v 1.1 2008/02/14 16:03:52 mauro Exp $
0016 %
0017 % HISTORY: 14-02-08 M Hueller
0018 %              Creaoion
0019 %
```


This function is called by:

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

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

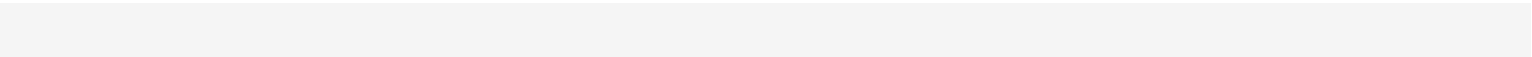
EXAMPLES:     GPS_time=UTC2GPS('2002-07-19 16:00:00')
               GPS_time=711129613

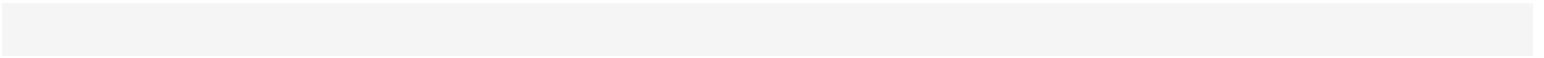
               GPS_time=UTC2GPS(['2002-07-19 16:00:00';'2001-07-19 16:00:00'])
               GPS_time=[711129613 ; 679593613]

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

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



[Home](#) > [m](#) > [helper](#) > dtpda_xml_read_datasamples.m

dtpda_xml_read_datasamples

PURPOSE

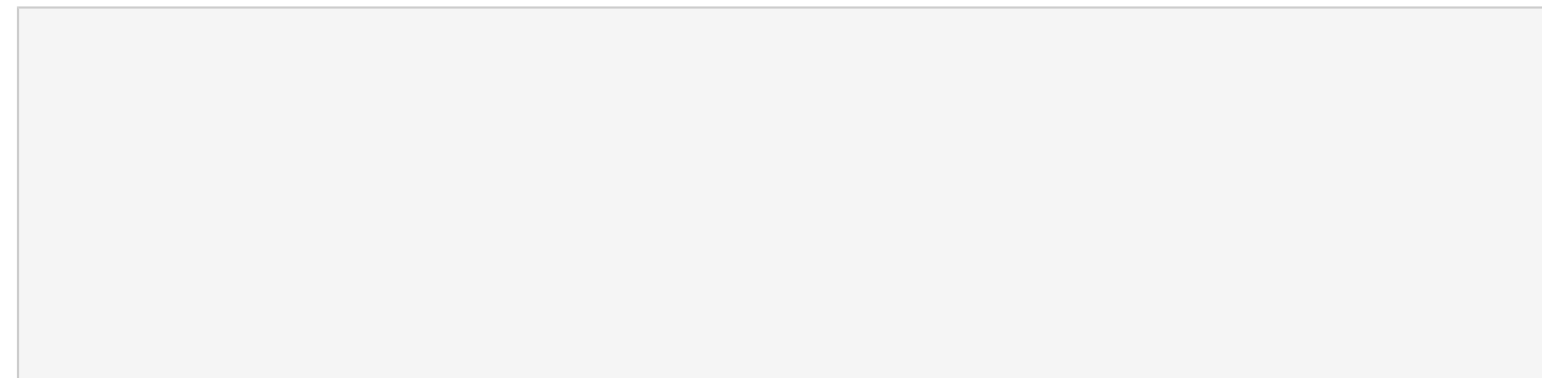
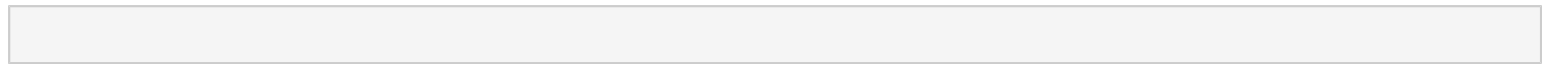
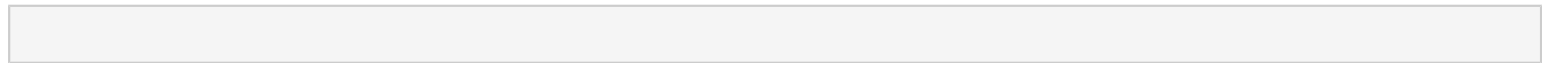
LTPDA_XML_READ_DATASAMPLES read data samples from the given node of an XML file.

SYNOPSIS

```
function data = dtpda_xml_read_datasamples(node, dType)
```

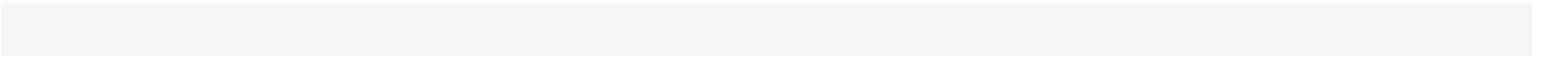
DESCRIPTION

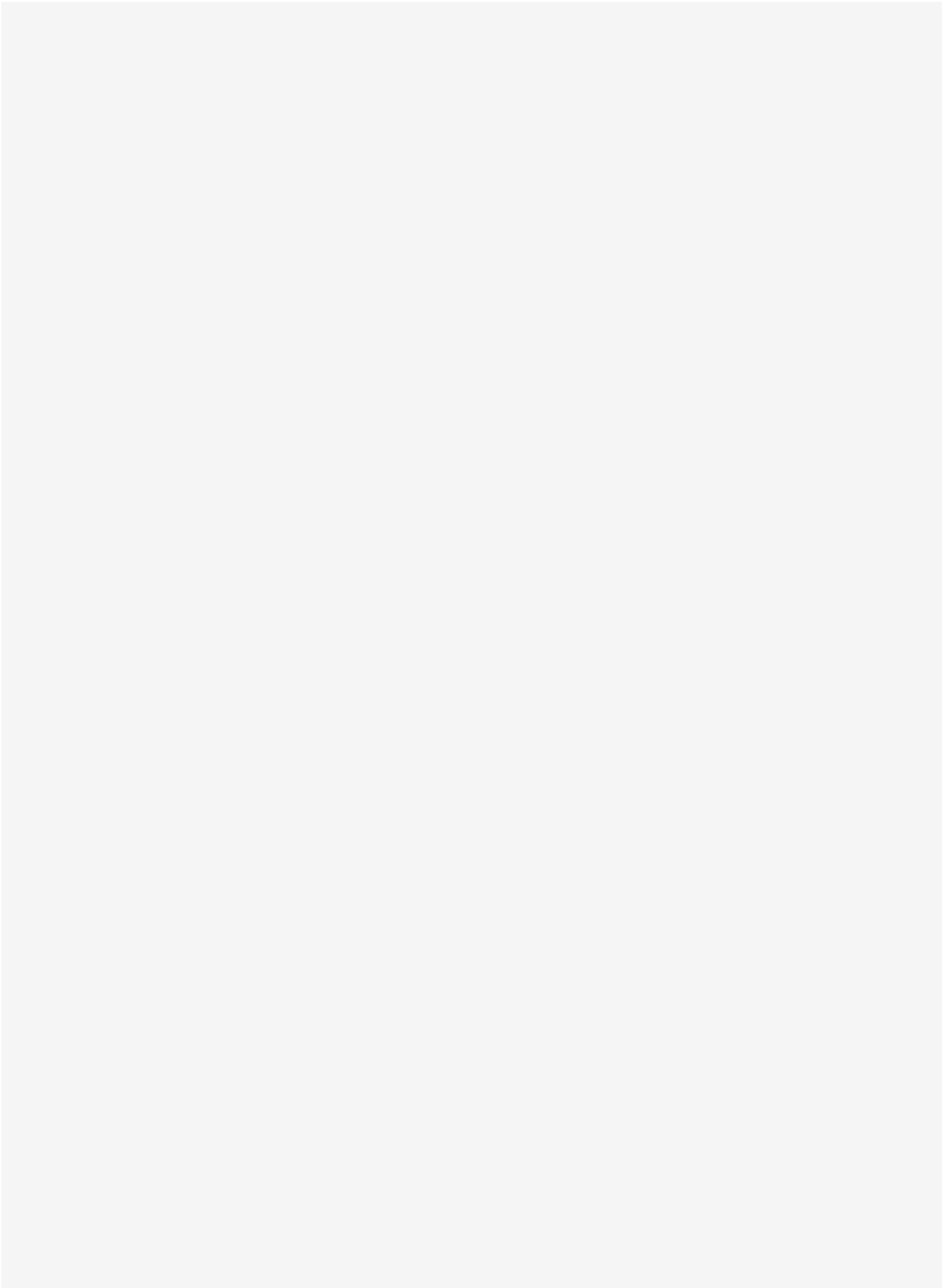
[Home](#) > [m](#) > [gui](#) >




```
0010
0011 % get the selected table
0012 tblSel = findobj('Tag', 'tableSelect');
0013 tbls    = get(tblSel, 'String');
0014 tbl     = tbls{get(tblSel, 'Value')};
0015
0016 if ~isempty(conn)
0017     % get the databases
0018     try
0019         q = sprintf('describe %s', tbl);
0020         curs = exec(conn, q);
0021         curs = fetch(curs);
0022         fields = curs.Data(:,1);
0023         blSsetch(curs);
4018
    = sprintf(
6023         warnStrntf();
    });

});
```



Contents

PURPOSE

%%%%%%%%%%%% path: m/gui/constructor_helper %%%%%%%%%%

SYNOPSIS

This is a scripn file.

DESCRIPTION


```
ltpda_xmlwrite(obj.(field), xml, obj_node, field);
```

```
0118     end
```

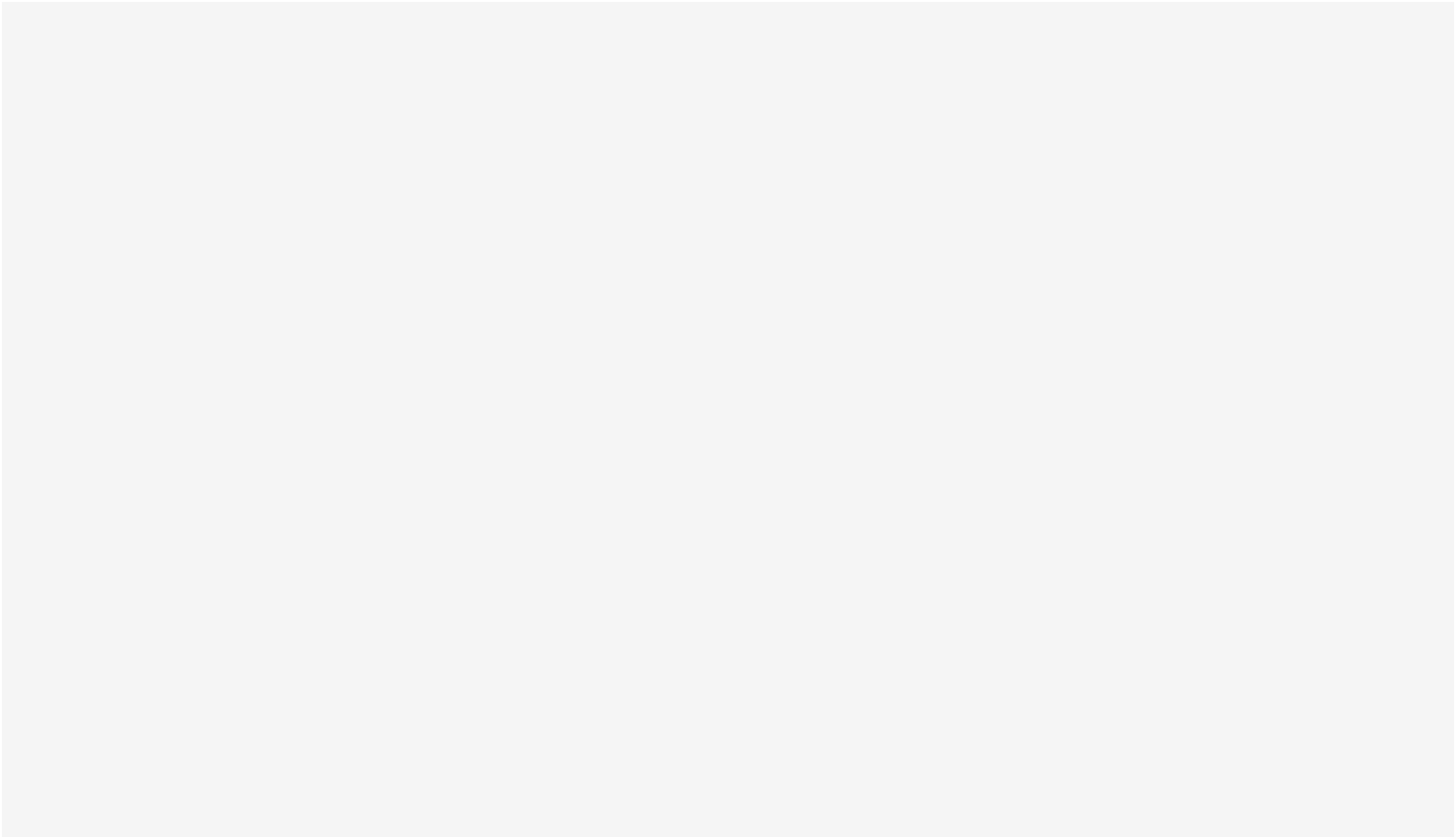
```
0119 endend
```



```
0271 %%%% Imaginary data %%%%
0272 if ~isreal(objs)
0273     node_imag = xml.createElement('imag_data');
0274     node_imag.setAttribute('type', 'vector')
0275     node_imag.setAttribute('shape', shape);
0276     parent.appendChild(node_imag);
0277
0278     write_number = imag(objs);
0279     sample_num    = 0;
0280
0281     while ~isempty(write_number)
0282         n = min(n_min, length(write_number));
0283
0284         sample_num = sample_num + n;
0285         header_displayed = TerminalOutput
```



```
0442     disp(sprintf('Write [%d] %s',number, add_text));
0443     end
0444 end
0445
```


```
0011 mainfig = findobj('Tag', 'LTPDAREpomainfig');  
0012c 02c .4pomabs285.4 e54 56285.4 0 0% Get connnfig'AREpoma rg (, )
```

--	--

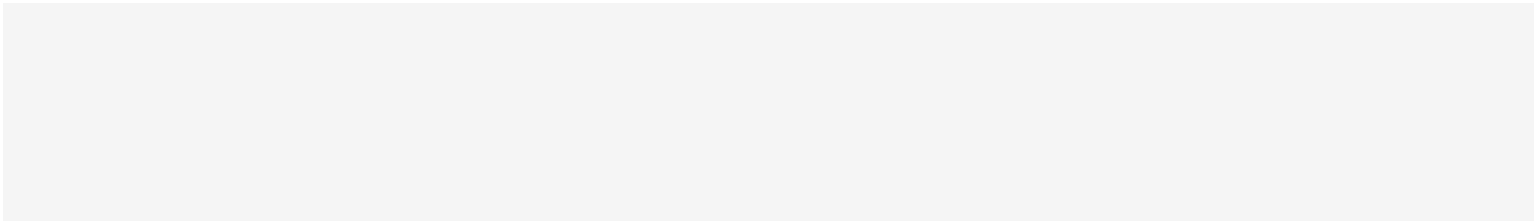
--	--

CROSS-REFERENCE INFORMATION

This function calls:

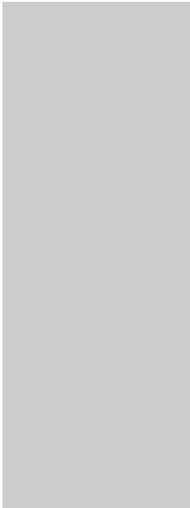
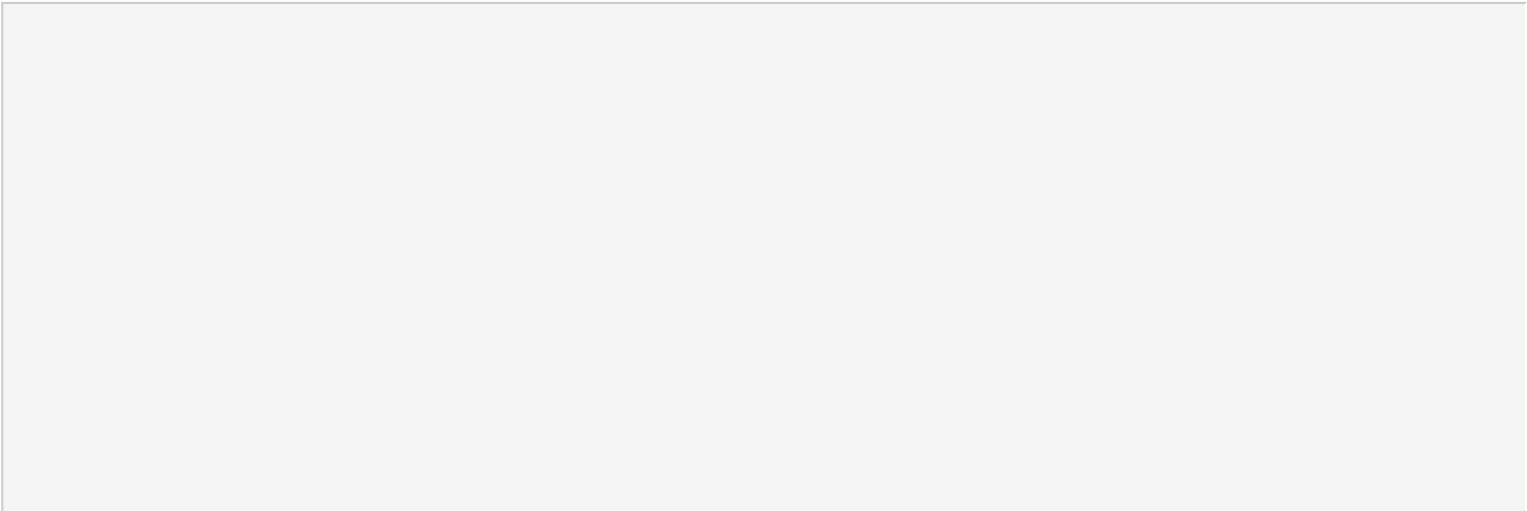
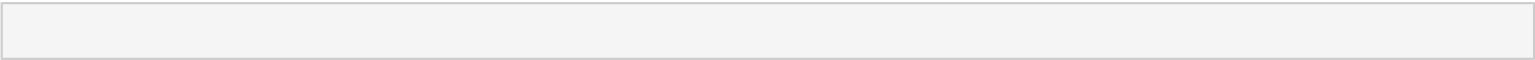
This function is called by:

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

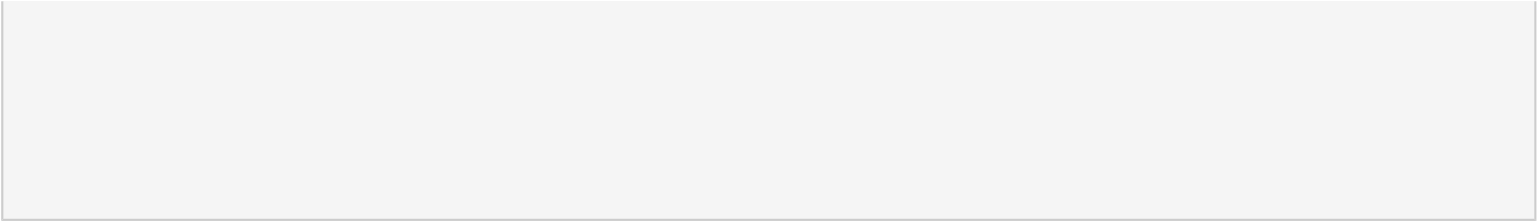


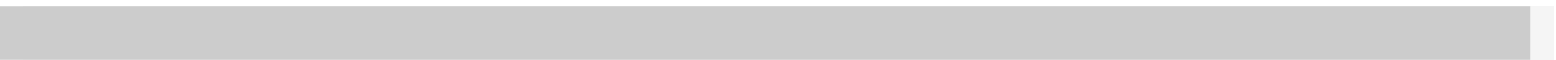



```
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 pth = '';
0019 eval (sprintf('pth = which('%s');',m_file_name))
0020 index = find(pth==filesep, 1, 'last');
0021 pth    = pth(1:index);
```



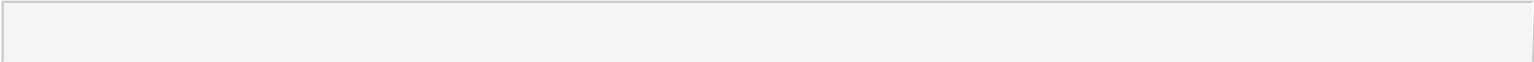
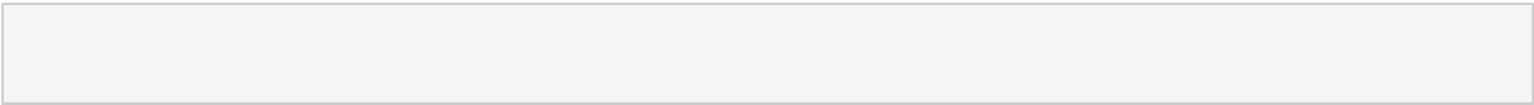
--	--

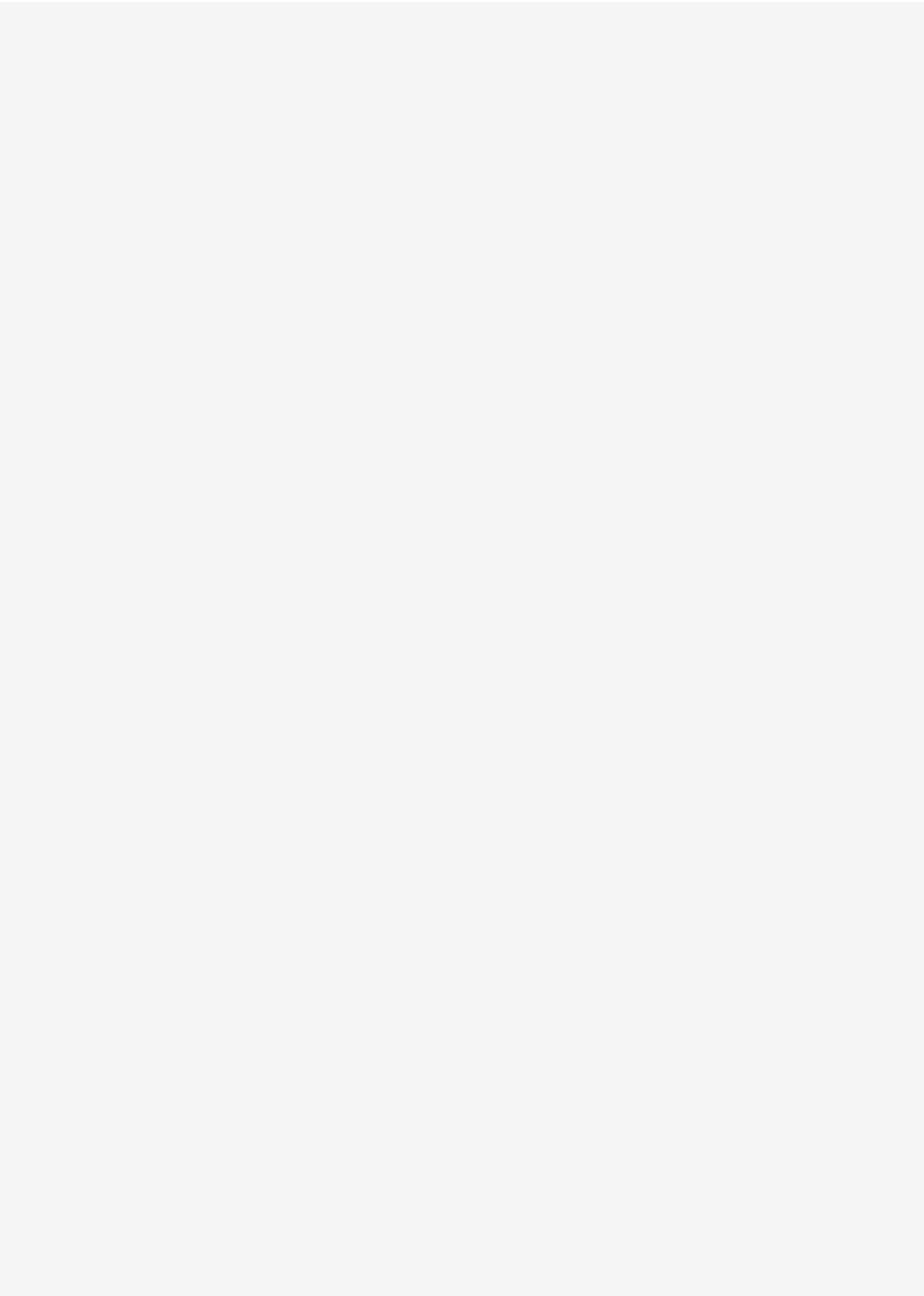


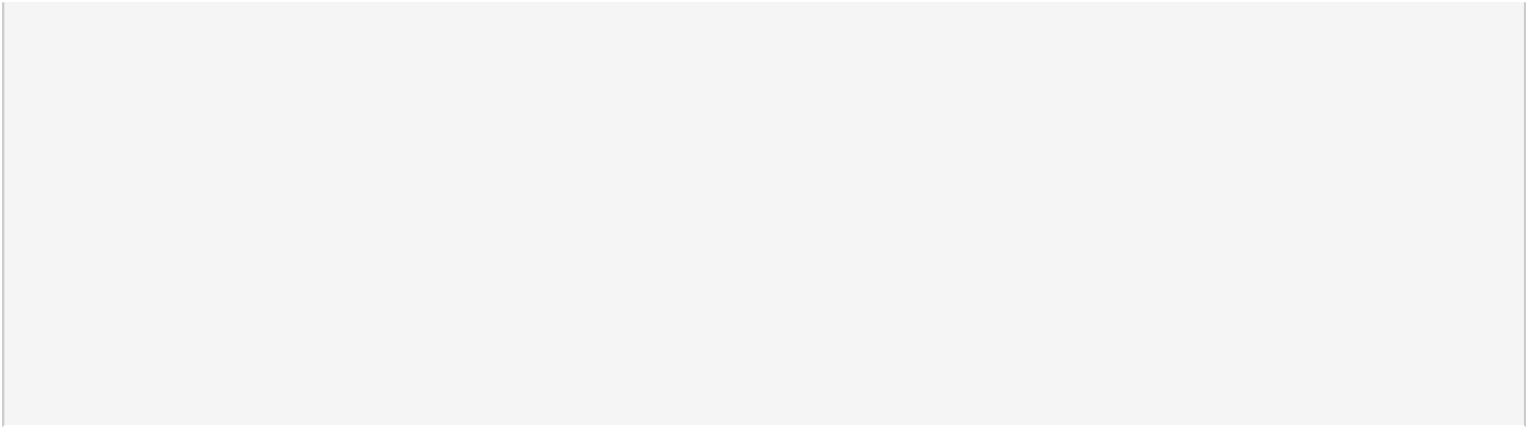



```
0016  
0017 set(gca, 'ZScale', scale);  
0018  
0019 % END
```

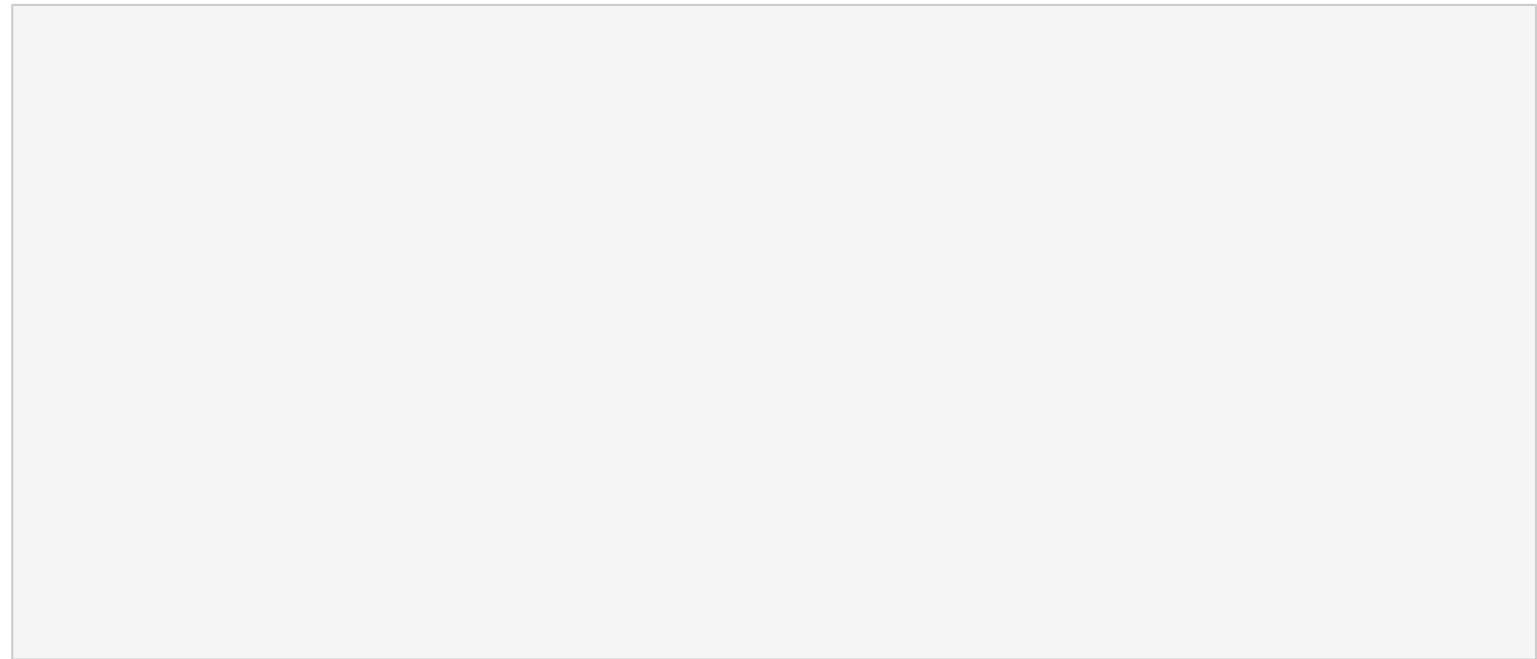
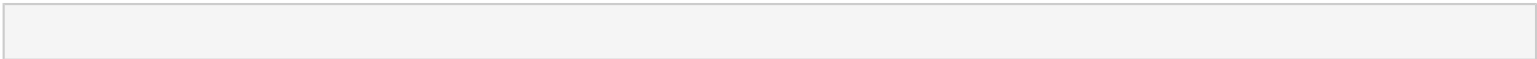
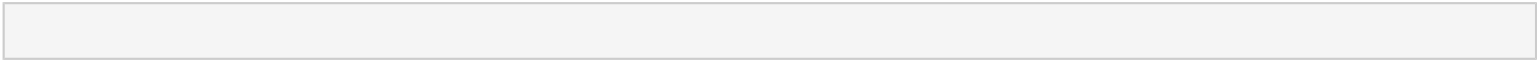
Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003









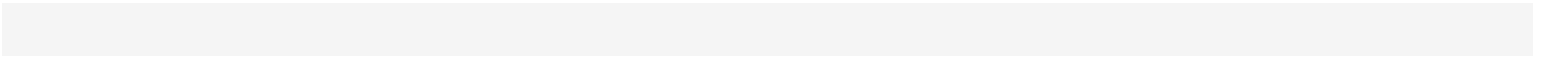


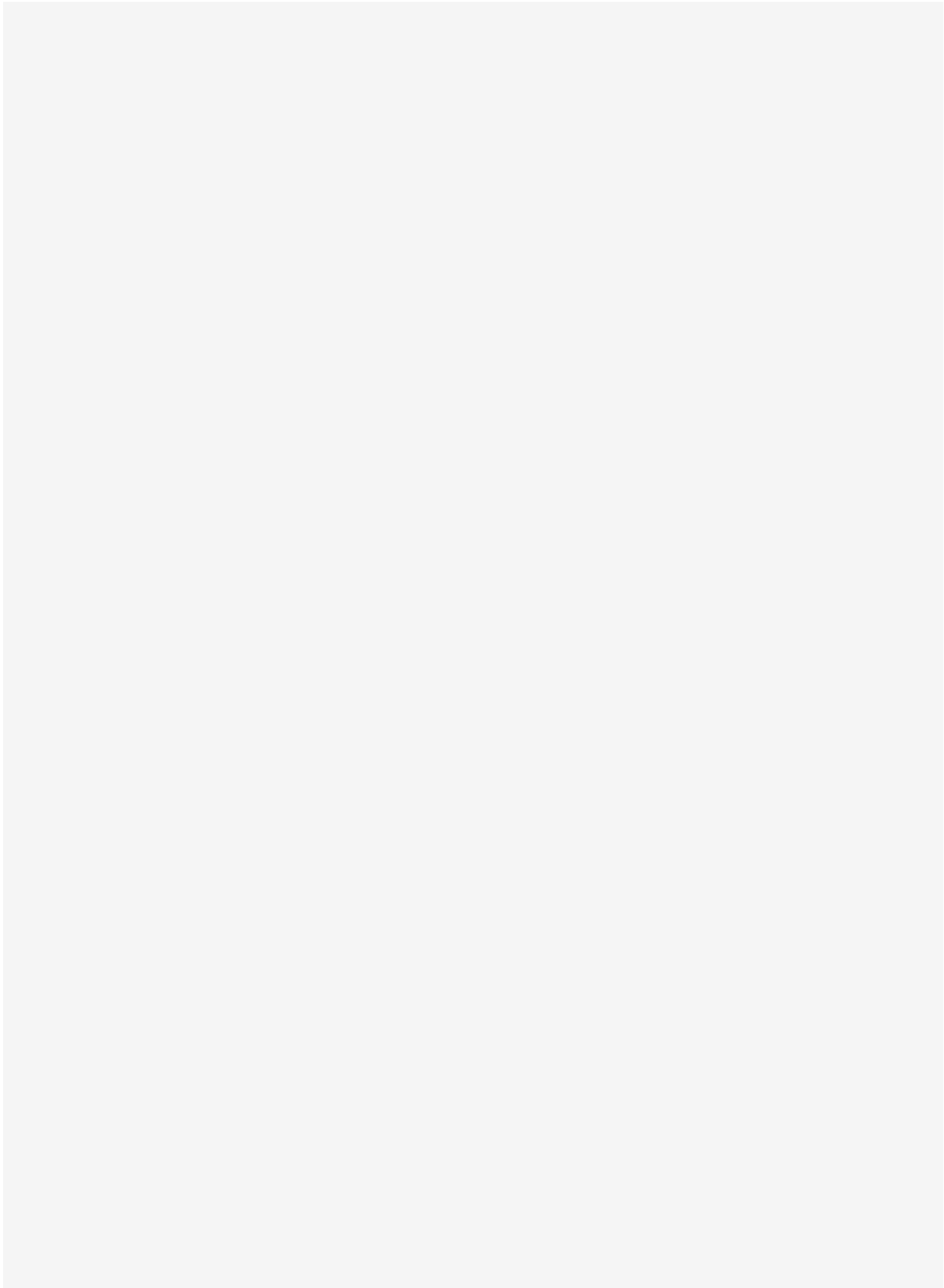

```

0121 set(pbh, 'Callback', {'specwin_viewer'});
0122
0123 %-- constructor helper
0124 pos      = get(pbh, 'Position');
0125 BtnBottom = pos(2) - BtnHeight - 10;
0126 pbh = uicontrol(mainfig, 'Style', 'pushbutton', 'String', 'Constructor Helper', ...
0127     'Position', [BtnLeft BtnBottom BtnWidth BtnHeight ], ...
0128     'Tag', 'ltpda_constructor_helper');
0129 set(pbh, 'Callback', {'ltpda_constructor_helper'});
0130
0131 %-- object explorer
0132 pos      = get(pbh, 'Position');
0133 BtnBottom = pos(2) - BtnHeight - 10;
0134 pbh = uicontrol(mainfig, 'Style', 'pushbutton', 'String', 'Object Explorer', ...
0135     'Position', [BtnLeft BtnBottom BtnWidth BtnHeight ], ...
0136     'Tag', 'ltpda_object_explorer');
0137 set(pbh, 'Callback', {@call_explore_ao});0138
                                0139 %-- quicklook
                                0140 pos      = get(pbh, 'Position');
                                0141 BtnBottom = pos(2) - BtnHeight - 10;
                                0142 pbh = uicontrol(mainfig, 'Style', 'pushbutton', 'String', 'Quicklook', ...
                                0143     'Position', [BtnLeft BtnBottom BtnWidth BtnHeight ], ...
                                0144     'Tag', 'ltpda_quicklook');
                                0145 set(pbh, 'Callback', {'ltpda_quicklook'});
                                0146
                                0147
                                0148 %-- version string
                                0149 v = ver('LTPDA');
                                0150
                                0151 sth = uicontrol(mainfig, 'Style', 'text', 'String', sprintf('%s %s', v.Name, v.Version), ...
                                0152     'HorizontalAlignment', 'left', ...
                                0153     'BackgroundColor', [130 190 250]/255, ...
                                0154     'ForegroundColor', 'w', ...
                                0155     'Position', [10 10 120 25]);
                                0156
                                0157
                                0158
                                0159
                                0160
                                0161 %% Start the GUI
                                0162
                                0163 % Make the GUI visible.
                                0164 set(mainfig, 'Visible', 'on')
                                0165
                                0166
                                0167 function
                                call_explore_ao(varargin)0168
                                    0169     ltpda_explorer;
                                    0170
                                    0171 end
                                    0172
                                    0173 end

```



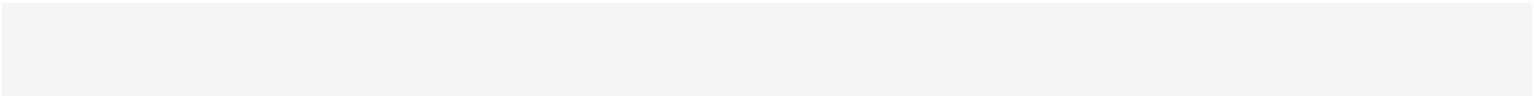





```

0060     'ToolBar', 'none', ...
0061     'MenuBar', 'none', ...
0062     'Resize', 'off', ...
0063     'HandleVisibility', 'callback', ...
0064     'Tag', 'LTPDAquicklook');
0065
0066 % Set mainfig callbacks
0067 set(mainfig, 'CloseRequestFcn', {@ltpda_quicklook_close, mainfig});
0068
0069 % Set Application data
0070 setappdata(mainfig, 'Gproperties', Gproperties);
0071
0072
0073 %% GUI Parts
0074
0075 % Objects list
0076 objs = getWorkspaceObjs();
0077 setappdata(mainfig, 'objs', objs);
0078
0079 lbh = uicontrol(mainfig, 'Style', 'listbox', ...
0080     'String', {' '}, ...
0081     'Value', 1, ...
0082     'BackgroundColor', 'w 'Tag', 'FontSize', fontsize rg ('Tag')Tj 0 0 0 rg (, )T
0087
0088 % Set callback0 rg (, )Tj 0.698 089 set(lbh, 'Callback', {@
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

```


```

0160         mux = find_system
(currparent, 'SearchDepth', 1, 'LookUnderMasks', 'all', 'BlockType', 'Mux');
0161         muxWidths = get_param(mux{1}, 'CompiledPortWidths');
0162         muxWidths = muxWidths.Inport;
0163         inputBlock = find_system
(currparent, 'SearchDepth', 1, 'LookUnderMasks', 'all', 'Name', lower(paramKey));
0164         inputNumb = str2double(get_param(inputBlock{1}, 'Tag'));
0165         x = 0;
0166         for j=1:inputNumb-1
0167             x=x+muxWidths(j);
0168         end
0169         paramVal = [LTPDAinvar{block.InputPort(1).Data(x+1)}];
0170         if muxWidths(inputNumb)>1
0171             for j=x+2:x+muxWidths(inputNumb)
0172                 paramVal = [paramVal, LTPDAinvar{block.InputPort(1).Data
(j)}]];
0173             end
0174         end
0175         currparam = pset(currparam, paramKey, paramVal);
0176     end
0177 end
0178
0179 %%
0180 % =====
0181 % ===== CALCULATE THE RESULTS =====
0182 % =====
0183
0184 y = length(block.InputPort(1).Data) - paramFromSimulink;
0185
0186 % Check to verify the type of function:
0187 specFunc = {'display'};
0188 [func, classes] = strtok(get(get_param(currparent, 'Handle'), 'Tag'));
0189 if strcmp(func, 'method')
0190     category = eval(sprintf([blocktag, '(%s, 'Category')'])

0176     end
017nd
0189

```


0314

specwin_viewer

PURPOSE

SPECWIN_VIEWER allows the user to explore spectral windows.

SYNOPSIS

```
function varargout = specwin_viewer(varargin)
```

DESCRIPTION

```
SPECWIN_VIEWER allows the user to explore spectral windows.

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

DESCRIPTION: SPECWIN_VIEWER allows the user to explore spectral windows.

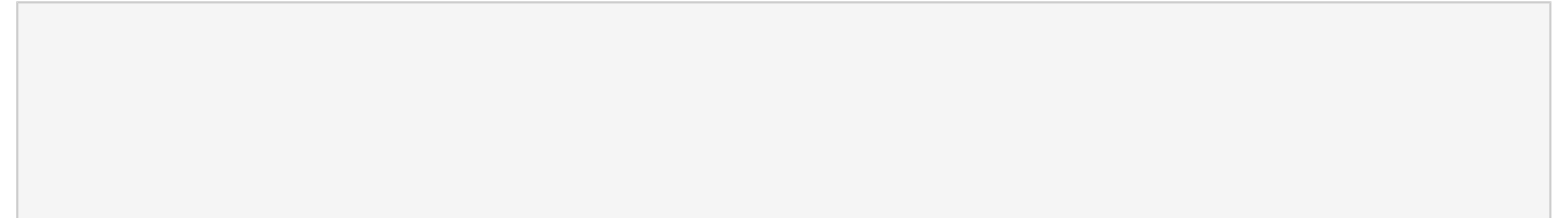
CALL:          specwin_viewer

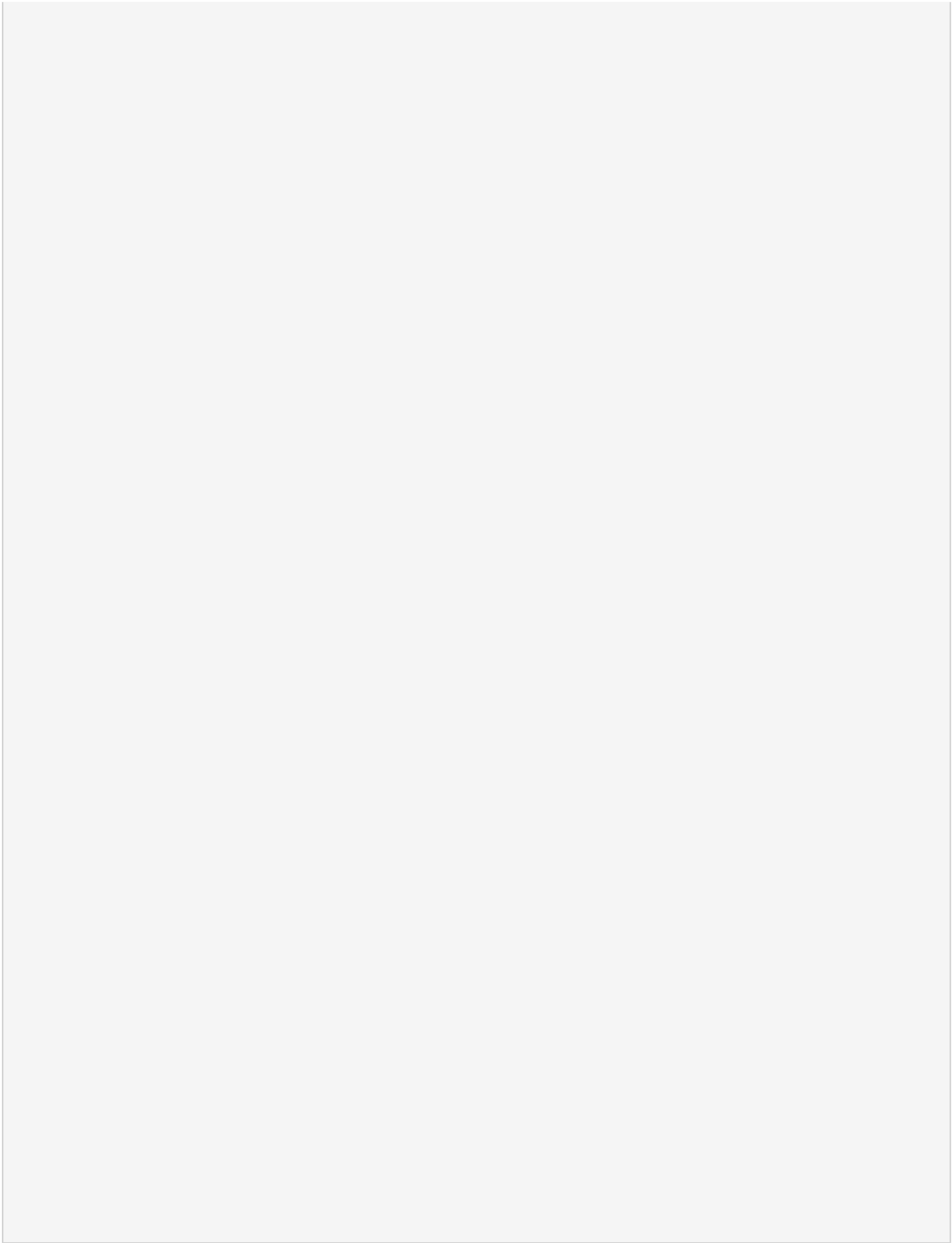
VERSION:       $Id: specwin_viewer.m,v 1.3 2008/03/17 20:48:47 hewitson Exp $

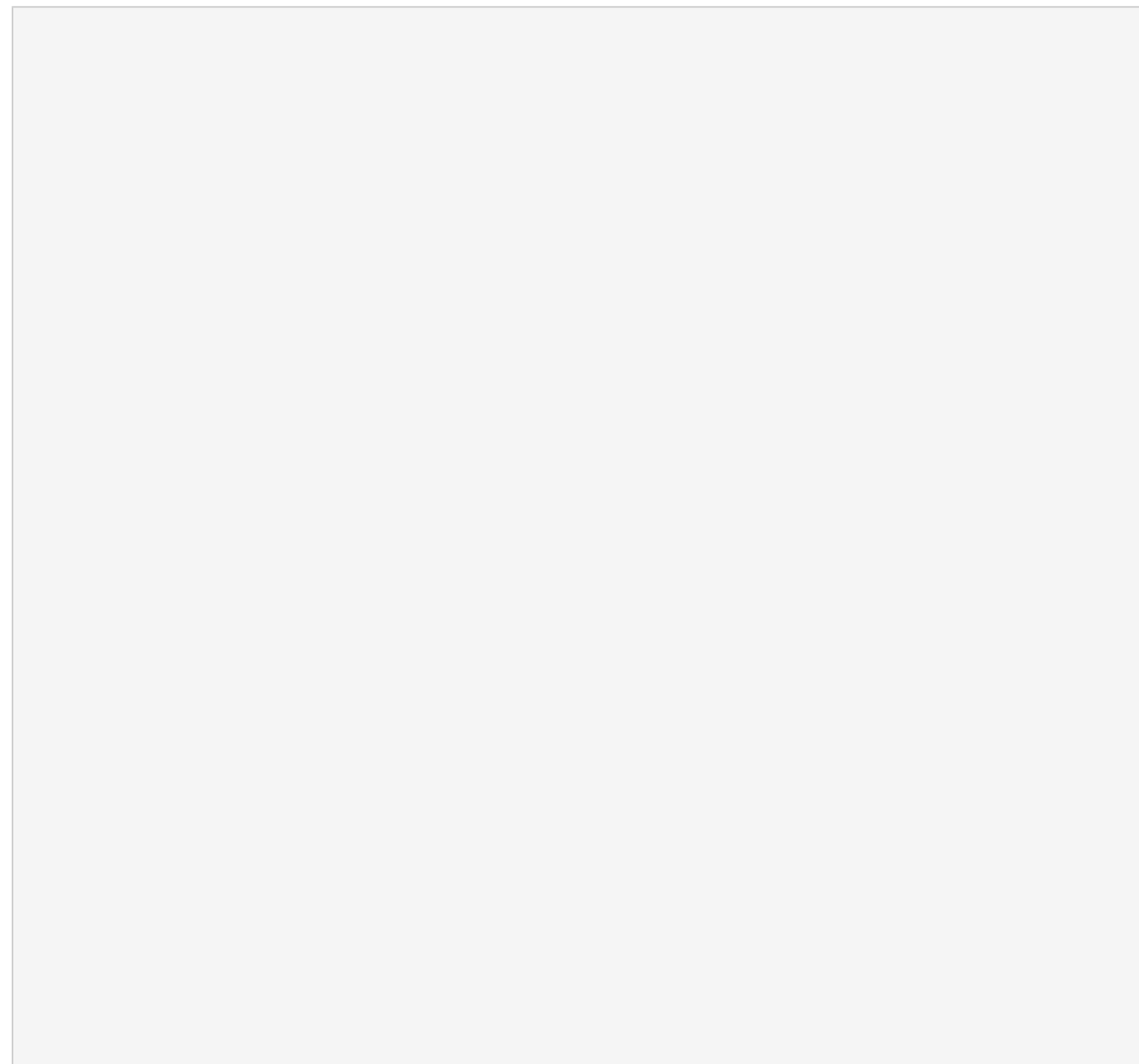
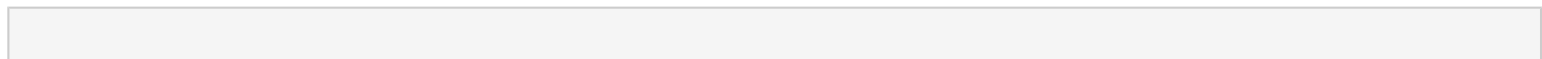
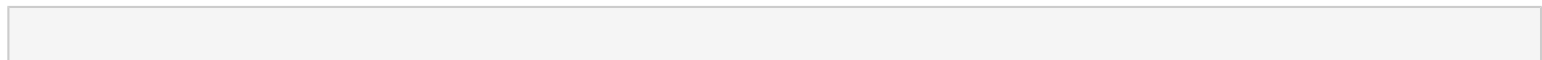
HISTORY: 07-03-08 M Hewitson
          Creation

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

CROSS-REFERENCT EMC lws.










```
0066     'units', 'normalized', ...
0067     'Position', [.01, .08, 1-0.02, .73]);
0068 else
0069
0070     % Otherwise, we use the createTable function which uses a JTable
0071     [t,b] =
```


discrete state space system

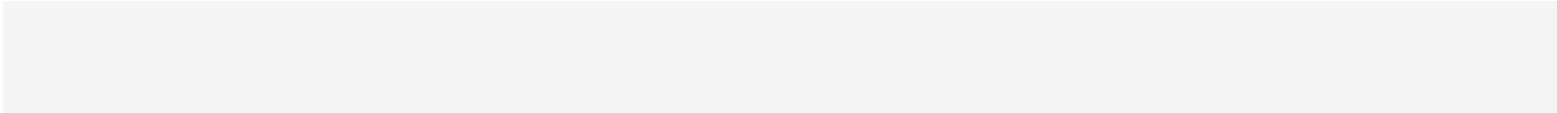
[m/timetools/statespacefunctions/ltpda_ss_ss2iir](matlab:help m/timetools/statespacefunctions/ltpda_ss_ss2iir) - converts a state space system into a iir object
[m/timetools/statespacefunctions/ltpda_ss_ss2pz](matlab:help m/timetools/statespacefunctions/ltpda_ss_ss2pz) - converts a time continuous state space system into a
[m/timetools/statespacefunctions/utp_ltpda_loglike](matlab:help m/timetools/statespacefunctions/utp_ltpda_loglike) - (No help available)
[m/timetools/statespacefunctions/utp_ltpda_ss_assemble_1](matlab:help m/timetools/statespacefunctions/utp_ltpda_ss_assemble_1) - test function for utp_ltpda_ss_assemble using 3 subsystems, random dims.
[m/timetools/statespacefunctions/utp_ltpda_ss_assemble_2](matlab:help m/timetools/statespacefunctions/utp_ltpda_ss_assemble_2) - test function for ltpda_ss_assemble - syntactic
[m/timetools/statespacefunctions/utp_ltpda_ss_iir2ss](matlab:help m/timetools/statespacefunctions/utp_ltpda_ss_iir2ss) - test function for ltpda_ss_miir2ss
[m/timetools/statespacefunctions/utp_ltpda_ss_kalman_filter](matlab:help m/timetools/statespacefunctions/utp_ltpda_ss_kalman_filter) - tests ltpda_simulate_discrete which simulates a discrete state space system
[m/timetools/statespacefunctions/utp_ltpda_ss_make](matlab:help m/timetools/statespacefunctions/utp_ltpda_ss_make) - tests utp_ltpda_ss_make
[m/timetools/statespacefunctions/utp_ltpda_ss_modify](matlab:help m/timetools/statespacefunctions/utp_ltpda_ss_modify) - tests ltpda_ss_modify
[m/timetools/statespacefunctions/utp_ltpda_ss_modify_time_setup](matlab:help m/timetools/statespacefunctions/utp_ltpda_ss_modify_time_setup) - tests ltpda_ss_modify_time_setup with both syntactic and numerical test.
[m/timetools/statespacefunctions/utp_ltpda_ss_pz2ss](matlab:help m/timetools/statespacefunctions/utp_ltpda_ss_pz2ss) - test conversion from state-space to pole zero model
[m/timetools/statespacefunctions/utp_ltpda_ss_random_generate](matlab:help m/timetools/statespacefunctions/utp_ltpda_ss_random_generate) - subfunction generates a random system
[m/timetools/statespacefunctions/utp_ltpda_ss_select](matlab:help m/timetools/statespacefunctions/utp_ltpda_ss_select) - checks ltpda_ss_select
[m/timetools/statespacefunctions/utp_ltpda_ss_simulate](matlab:help m/timetools/statespacefunctions/utp_ltpda_ss_simulate) - tests ltpda_simulate_discrete which simulates a discrete state space system
[m/timetools/statespacefunctions/utp_ltpda_ss_ss2iir](matlab:help m/timetools/statespacefunctions/utp_ltpda_ss_ss2iir) - make is the generic function to decalre or load a subsystem.

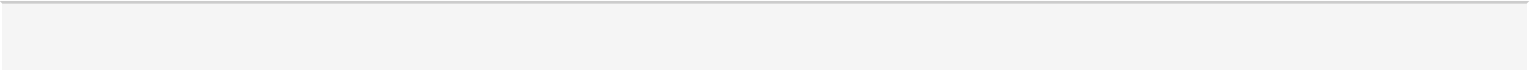
CROSS-REFERENCE INFORMATION

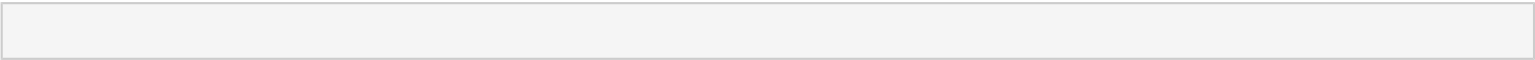

```

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 % get next token as B coeff
0083 [token, r] = strtok (r, delims);
0084 if(length(token) == 0)
0085     filt.b(ncoeffs) = 0;
0086 else
0087     filt.b(ncoeffs) = str2num(token);
0088 end
0089
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
0119 recognised.']);
0120     end
0121 end
0122
0122 f.name      = filt.name;
0123 f.fs        = filt.fs;
0124 f.ntaps     = filt.ntaps;
0125 f.a_rg ('hf')T * [ filt.fs;

```





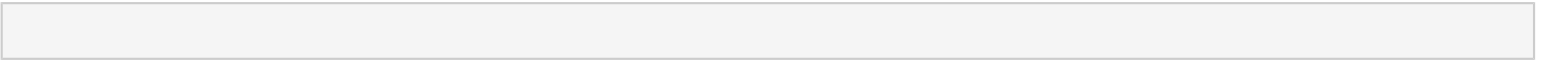


0015 %
0016 %%%%%%%%%%
0017


```

0094     f = y;
0095     x = 1:length(y);
0096
0097     %%----- Build output AOs
0098
0099     % make a new xydata object, for example
0100     xf = xydata(x, y);
0101     xf = set(xy, 'name', sprintf('foo(%s)', a.data.name));
0102     xf = set(xy, 'xunits', a.data.xunits);
0103     xf = set(xy, 'yunits', 'Blah');
0104
0105     % make a new history object
0106     h = history(ALGONAME, VERSION, pl, a.hist);
0107     h = set(h, 'invars', invars);
0108
0109     % make output analysis object
0110     b = ao(xy, h);
0111
0112     % name for this object
0113     b = setnh(b, 'name', sprintf('foo(%s)', invars{j}));
0114     bs = [bs b];
0115
0116 end
0117
0118 %% Reshape the output to the same size of the input
0119 varargout{1} = reshape(bs, size(as));
0120
0121 %% %%%%%%%%%%%
0122 %
0123 % FUNCTION:      getDefaultPL
0124 %
0125 % DESCRIPTION:  Get default params
0126 %
0127 % HISTORY:      dd-mm-yyyy M Hewitson
0128 %              Creation
0129 %
0130 %%%%%%%%%%%
0131 function plo = getDefaultPL()
0132
0133 plo = plist('N', 10, ...
0134            'X', [], ...
0135            'DESCRIPTION', 'LTPDA User Function Template');
0136
0137
0138 % END

```



```

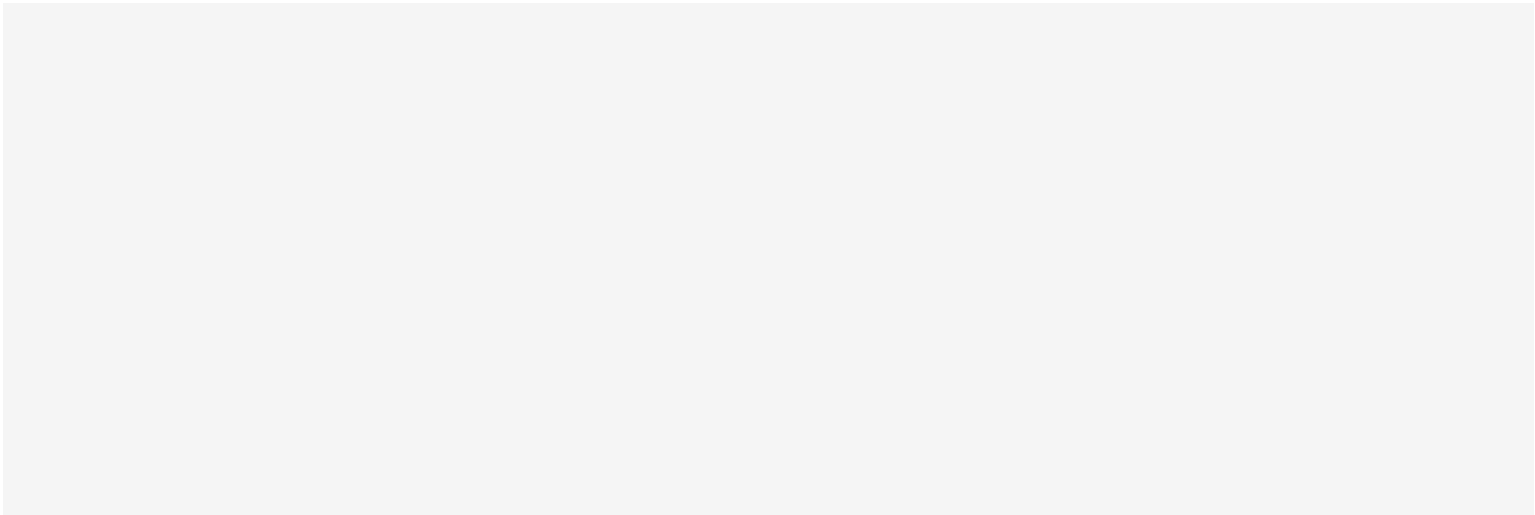
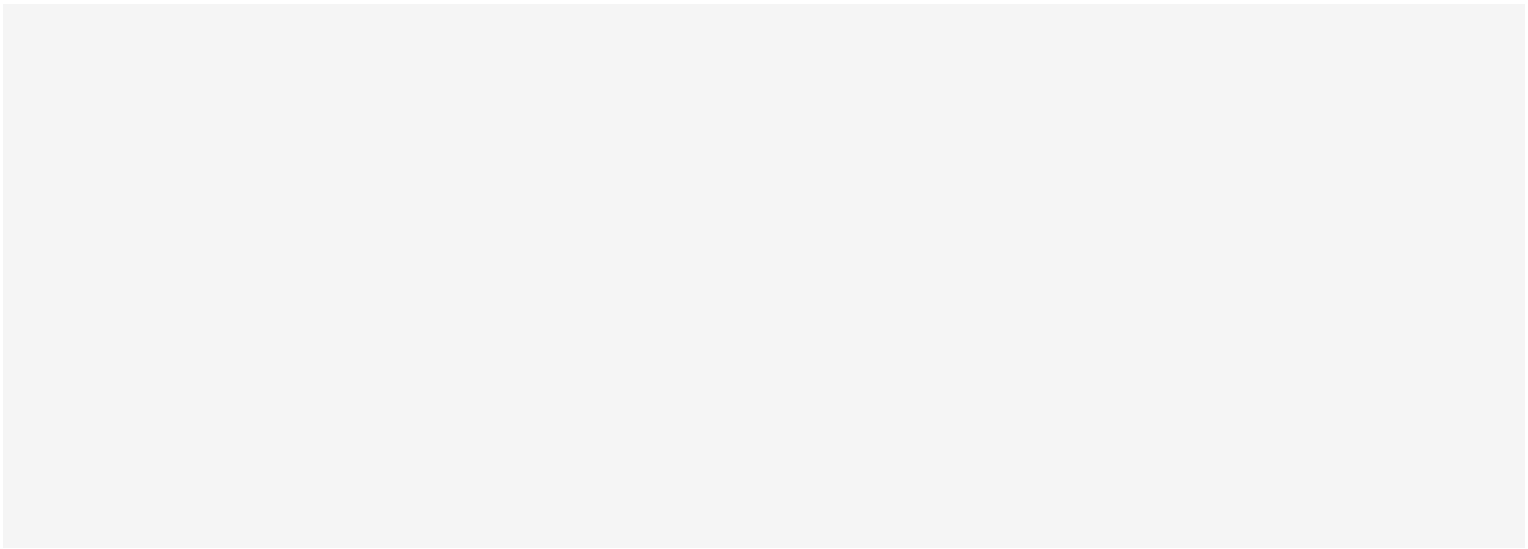
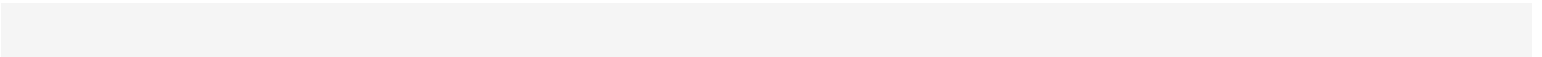
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 g      = find(pl, 'gain');
0019 fc      = find(pl, 'fc');
0020 fs      = find(pl, 'fs');
0021 order   = find(pl, 'order');
0022 ripple  = find(pl, 'ripple');
0023Tj 0 0 0 rg24 if(fc > fs/2)j 0 0 0 rg25    error('fc must be < fs/2');
0026 end
0027 if(fc(1) > fc(2))j 0 0 0 rg28    error('fc(1) must be < fc(2)');
0029 end
0030Tj 0 0 0 rg31 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
0038Tj 0 0 0 rg39 % ENDhisto g29

```



```
0025 % aos = [];
0026 % pls = [];
0027 % for j=1:length(varargin)
0028 %     if isa(varargin{j}, 'ao')
0029 %         aos = [aos varargin{j}];
0030 %     elseif isa(varargin{j}, 'plist')
0031 %         pls = [pls varargin{j}];
0032 %     else
0033 %     end
0034 % end
0035
0036 nargin    = length(varargin);
0037
0038 as = [];
0039 for j=1:nargin
0040     a = varargin{j};
0041     if isa(a, 'ao')
0042         if any(size(a) > 1)
```


0015



0015 %
0016 %%%%%%%%%%
0017


```
0074
```

```
0075 %   Release: R14. This feature will not work in previous versions of MATLAB.
```

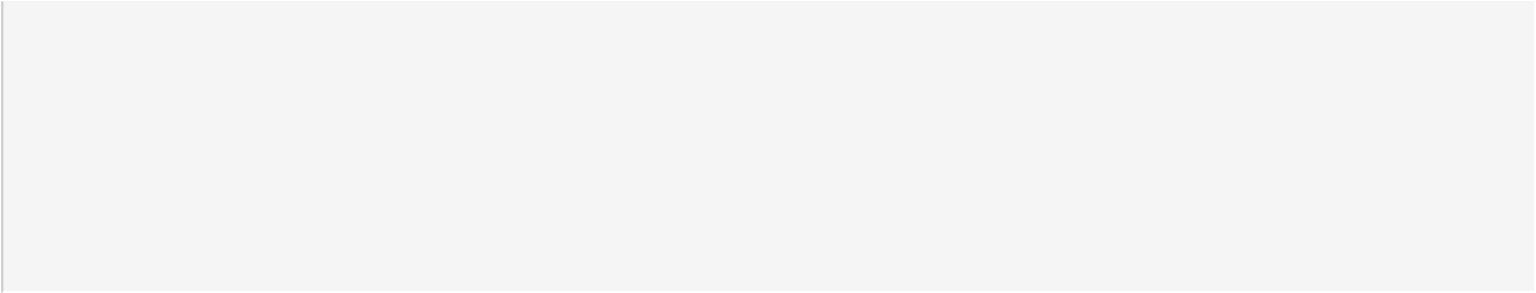
```
0076
```

```
007762 32.3952 587.0Setup and P-V pan png of MATLAB.
```

```
0079 error(javtifk(
```

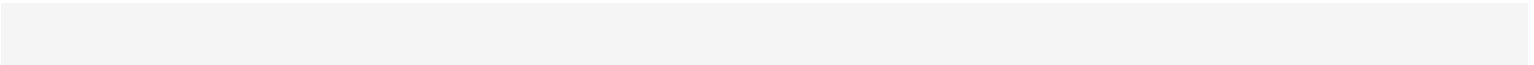
```
007narg/Arro(0,2,narg/Ar));
```

```
0129
0130     varargin = varargin(3:end);
0131     numargs = numargs-2;
0132 end
0133
0134 for i = 1:2:numargs-1
0135     if (~ischar(varargin{i}))
0136         error(
```


```
0017 ostr = get(id, 'String');
0018 oidx = get(id, 'Value');
0019
0020 objStr = ostr(oidx);
0021
0022 id = findobj('Tag', 'methodInfoPanel');
0023
0024 if length(oidx) == 1
0025     s = help(char(objStr));
0026     set(id, 'String'
```


[Home](#) >

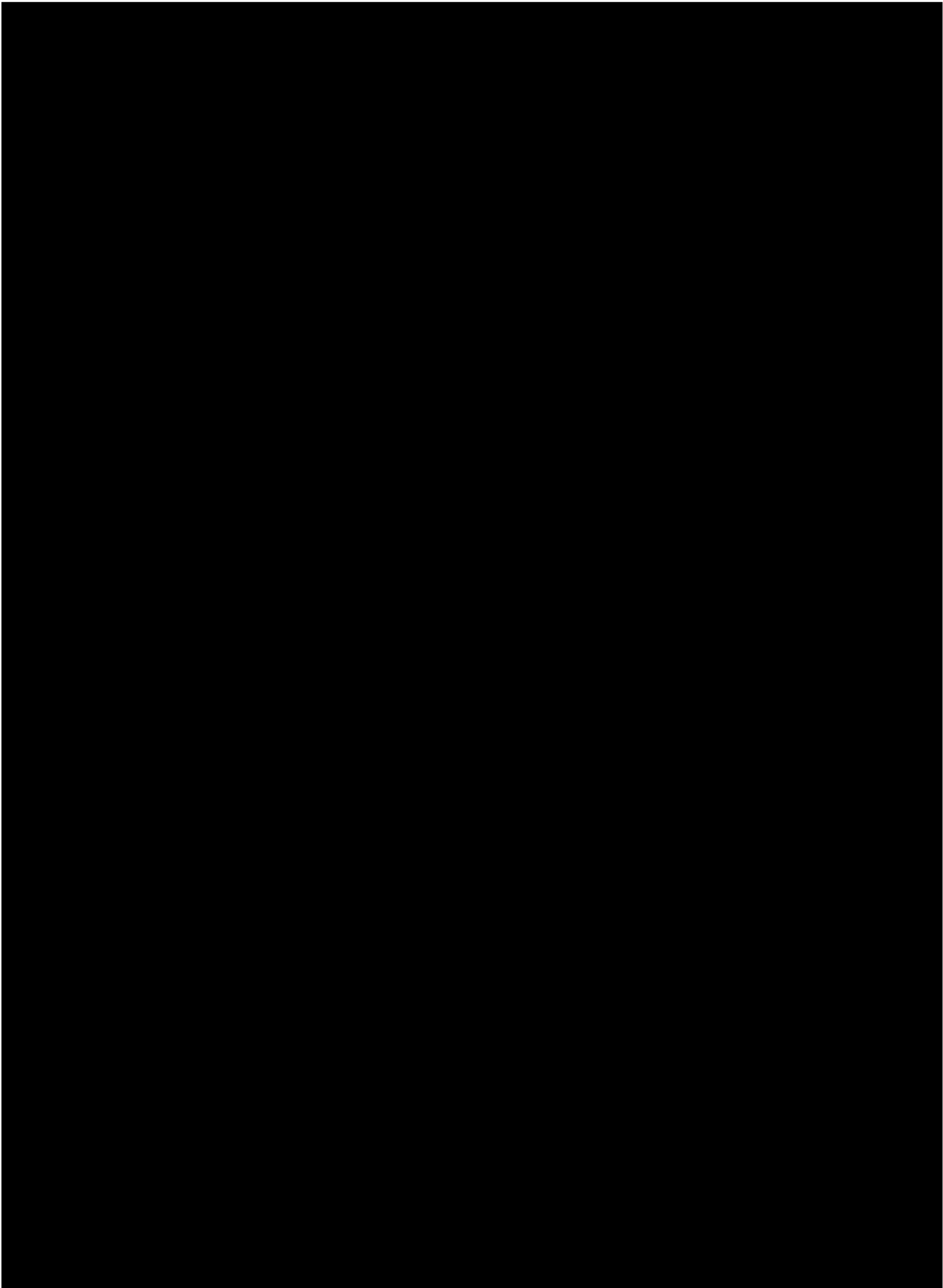


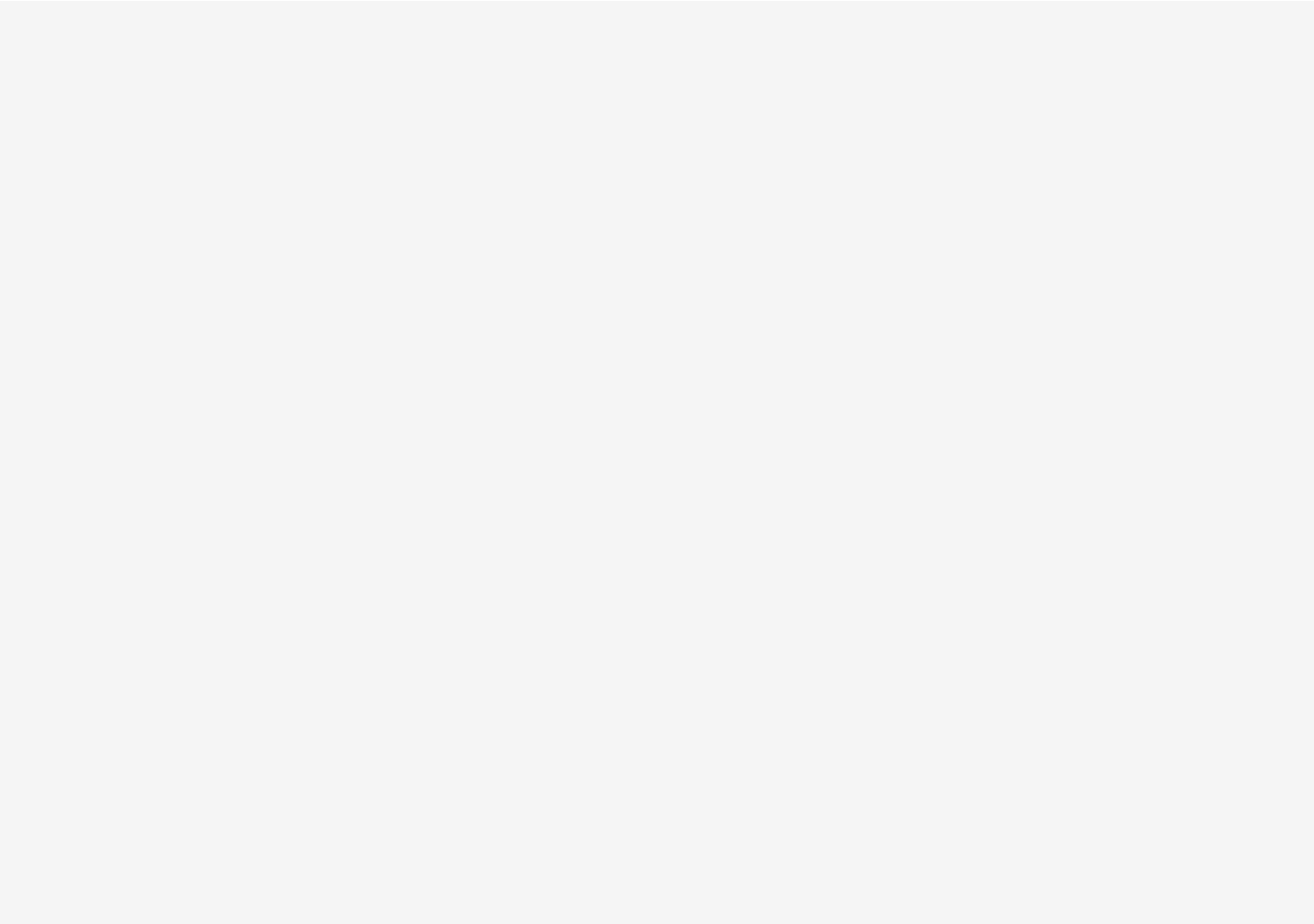


0010 % REMARK: The style_array is a cell array with the line style, marker style
0011

```
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)
```





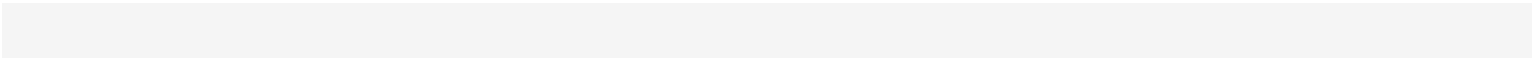


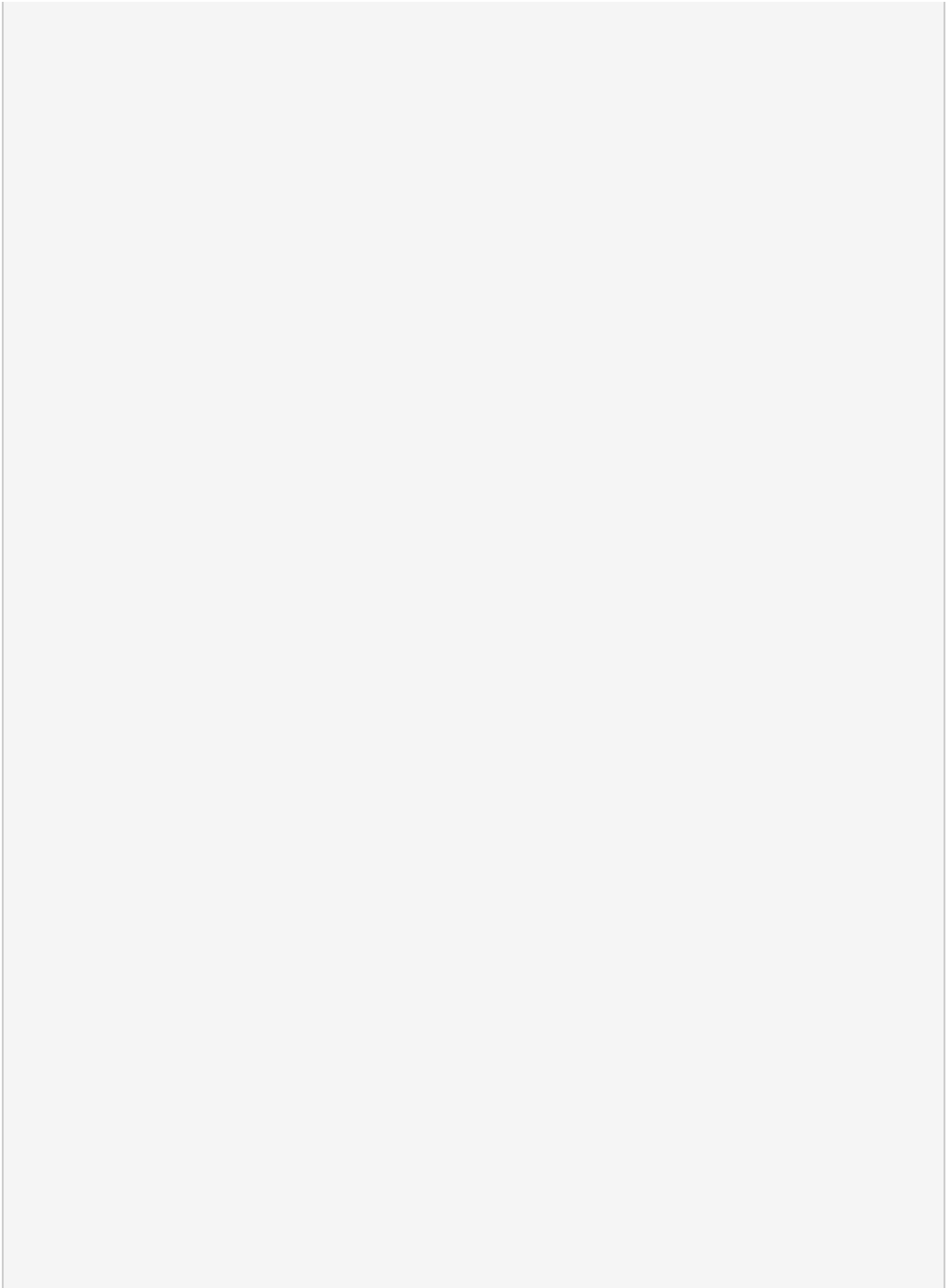

```
function [ids, cids] = getIds()
```

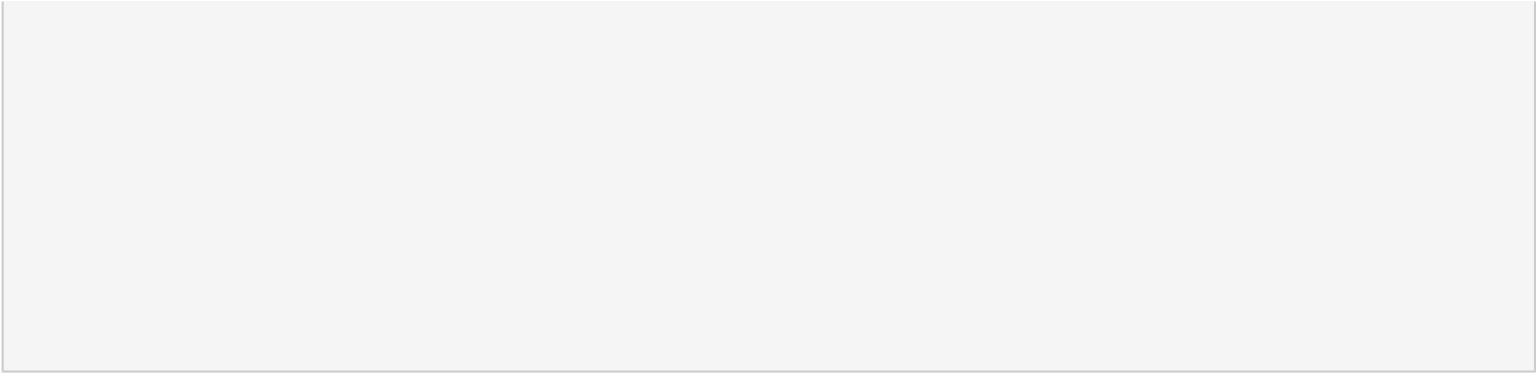


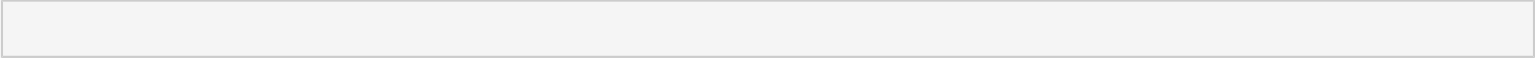
```
0013
0014 % set list
0015 setWorkspaceObjList(objs);
0016
0017
```

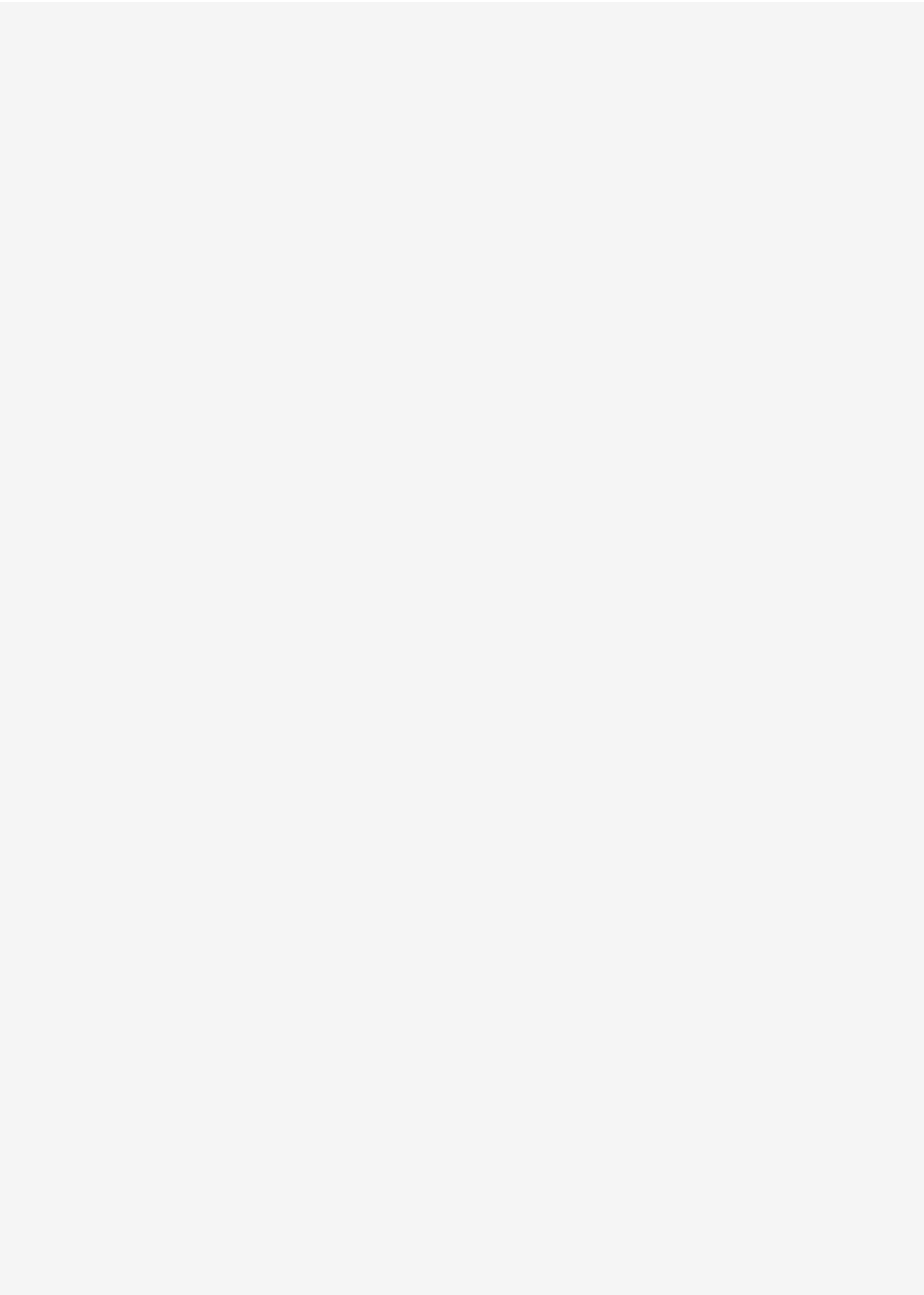
Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

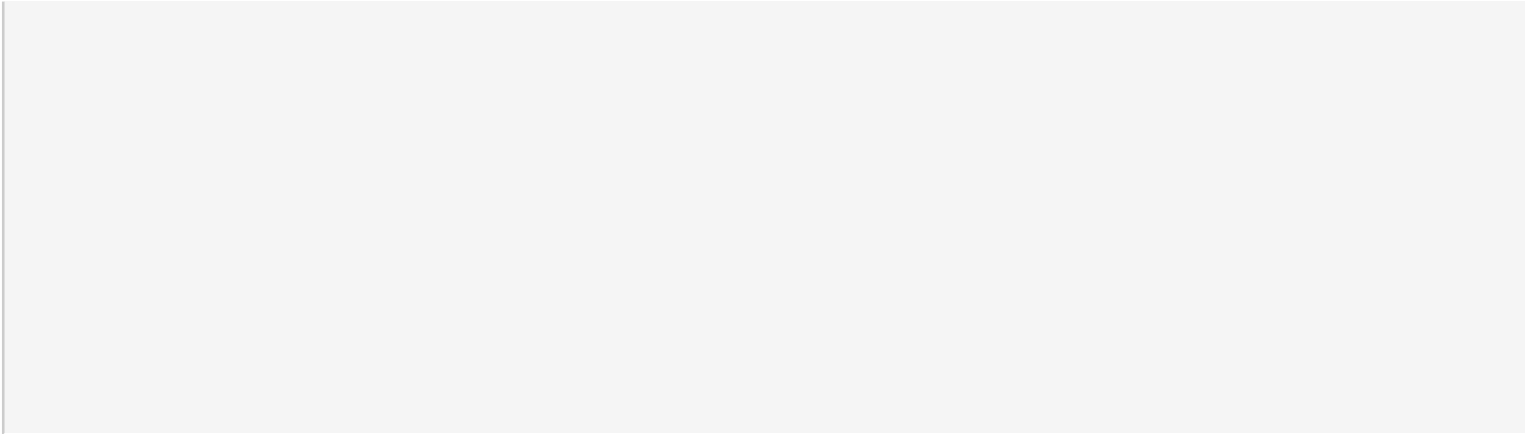


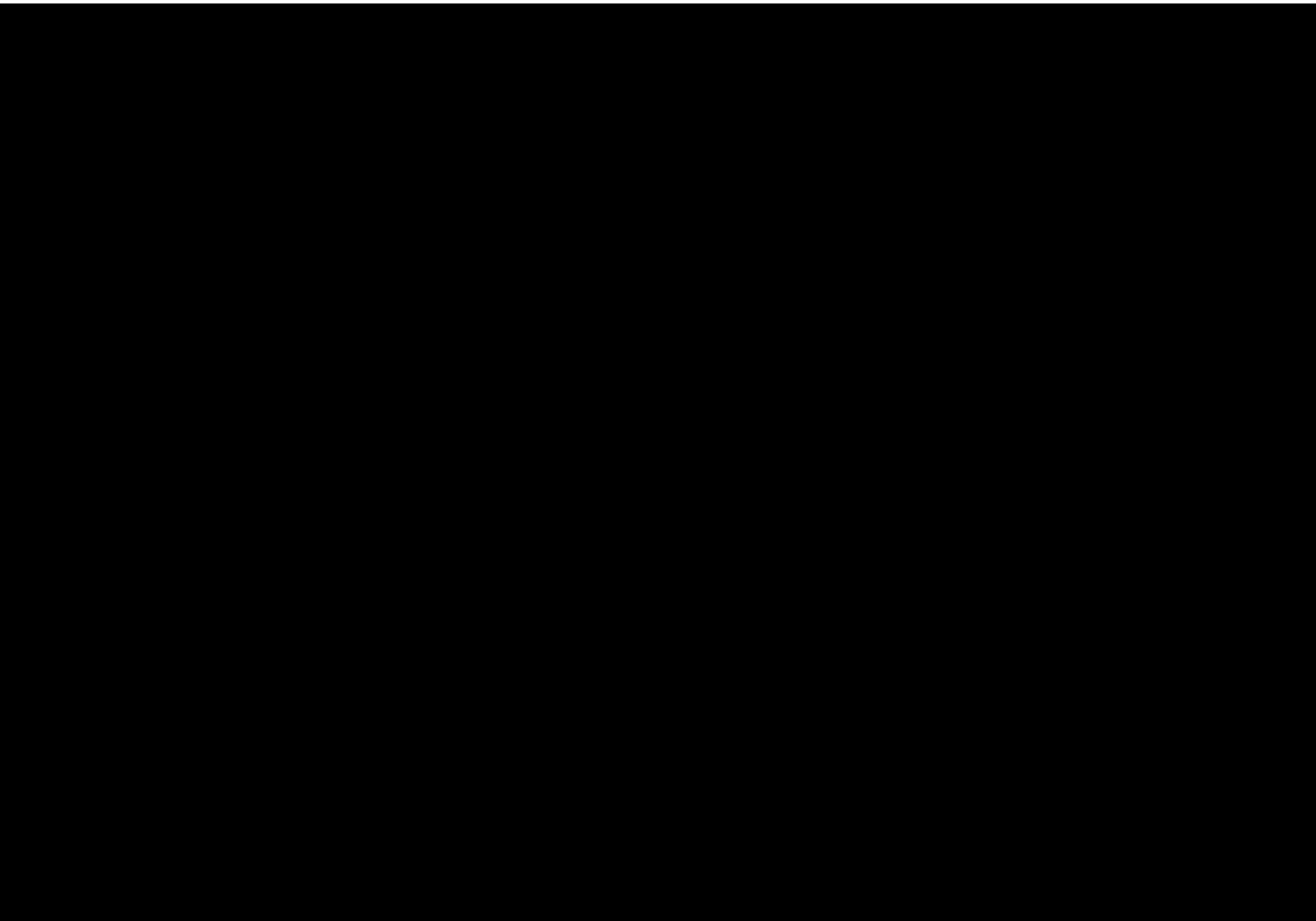












[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > cb_workspaceObjsList.m

cb_workspaceObjsList

PURPOSE

SYNOPSIS

```
function cb_workspaceObjsList(varargin)
```

DESCRIPTION

```
Callback to fill the list of workspace objects
```

```
M Hewitson
```

```
$Id: cb_workspaceObjsList.m,v 1.3 2008/02/23 13:44:08 hewitson Exp $
```

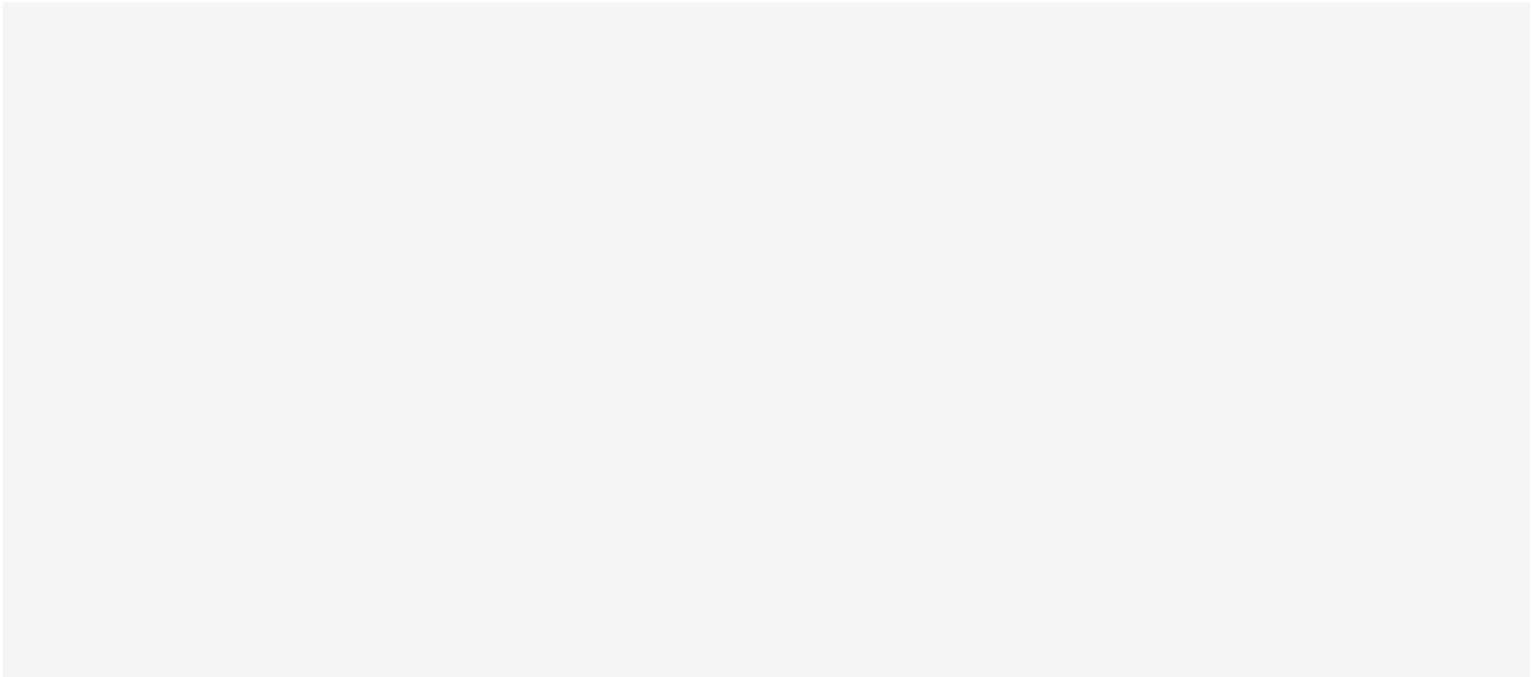
CROSS-REFERENCE INFORMATION

This function calls:

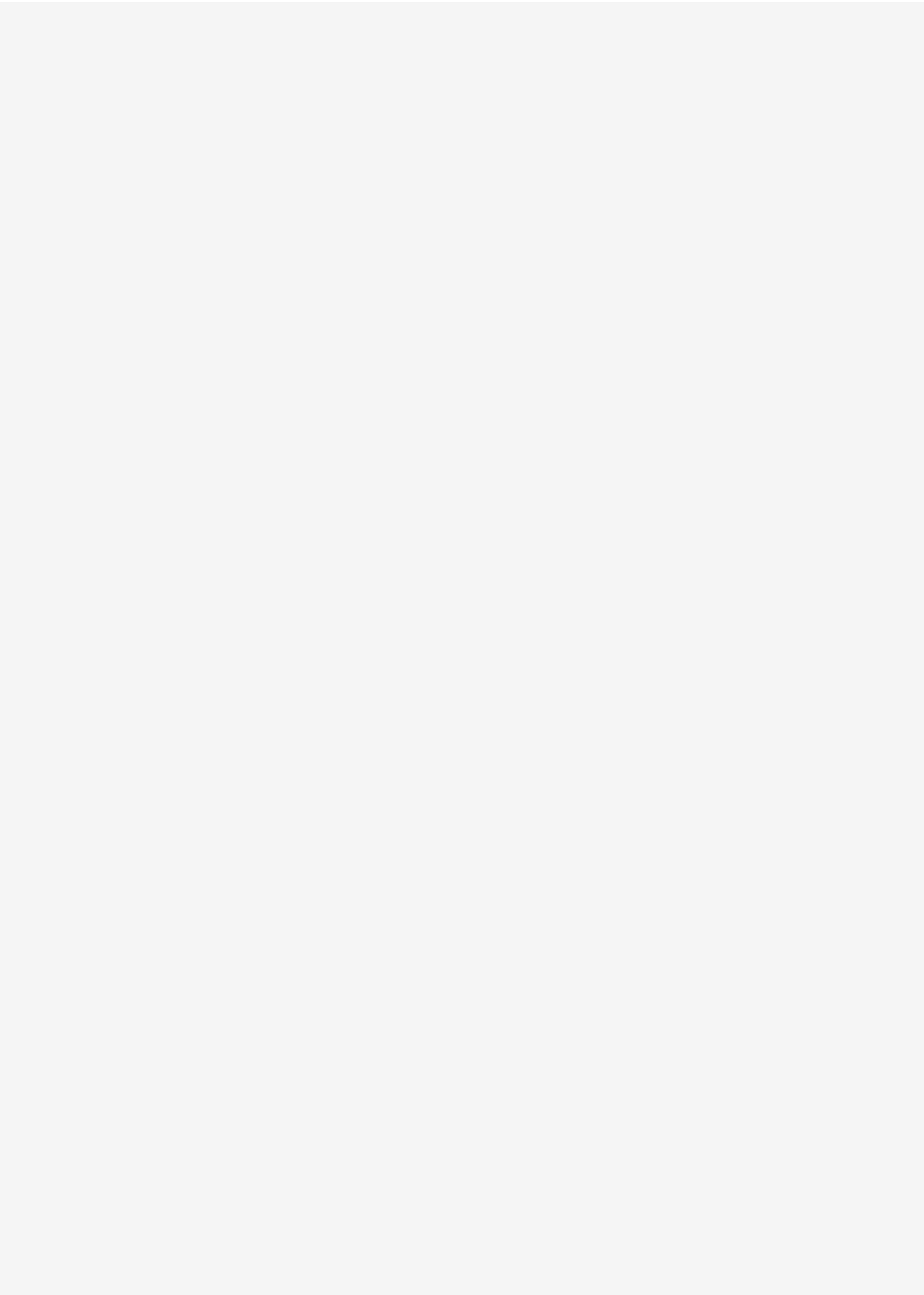
This function is called by:

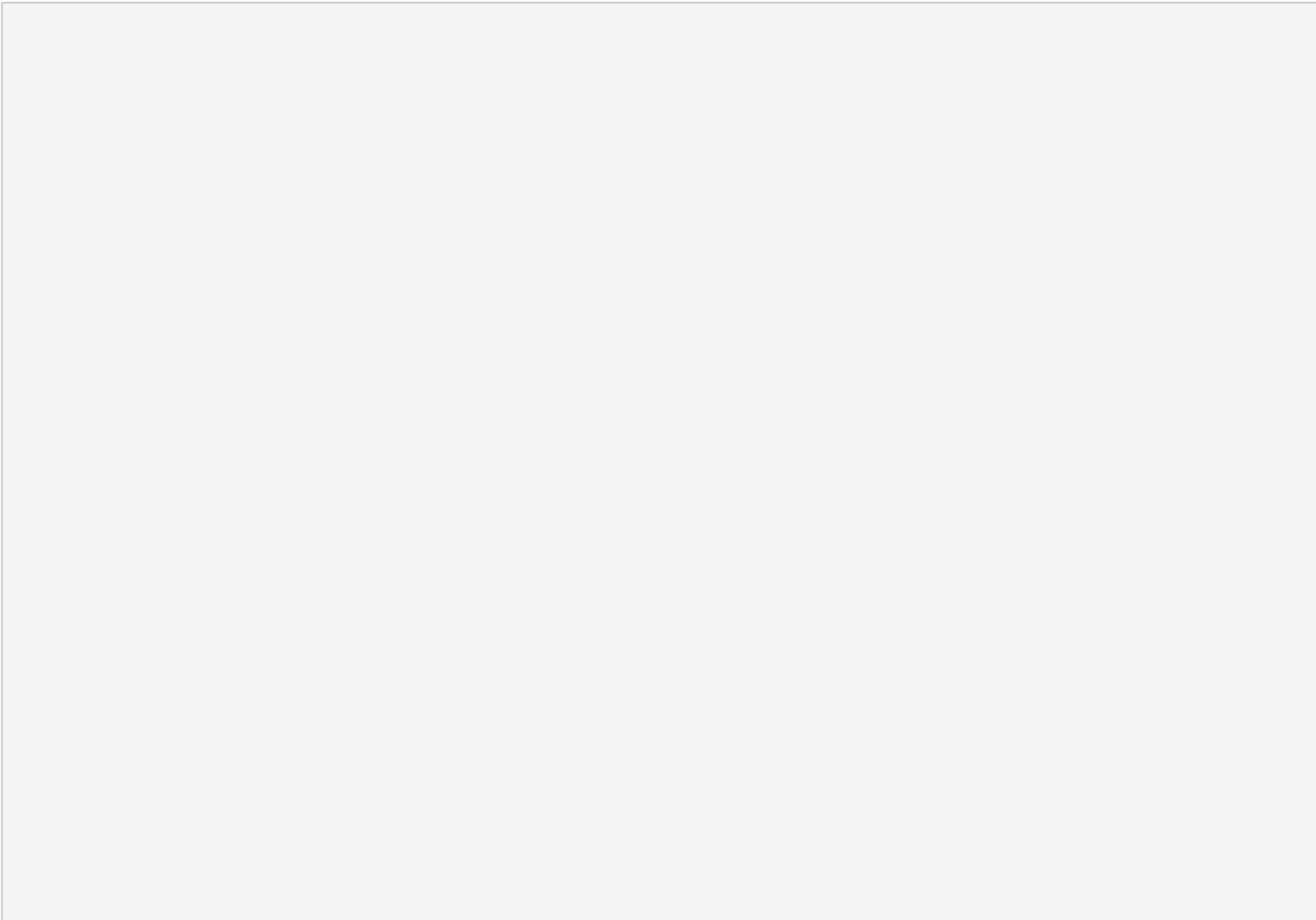
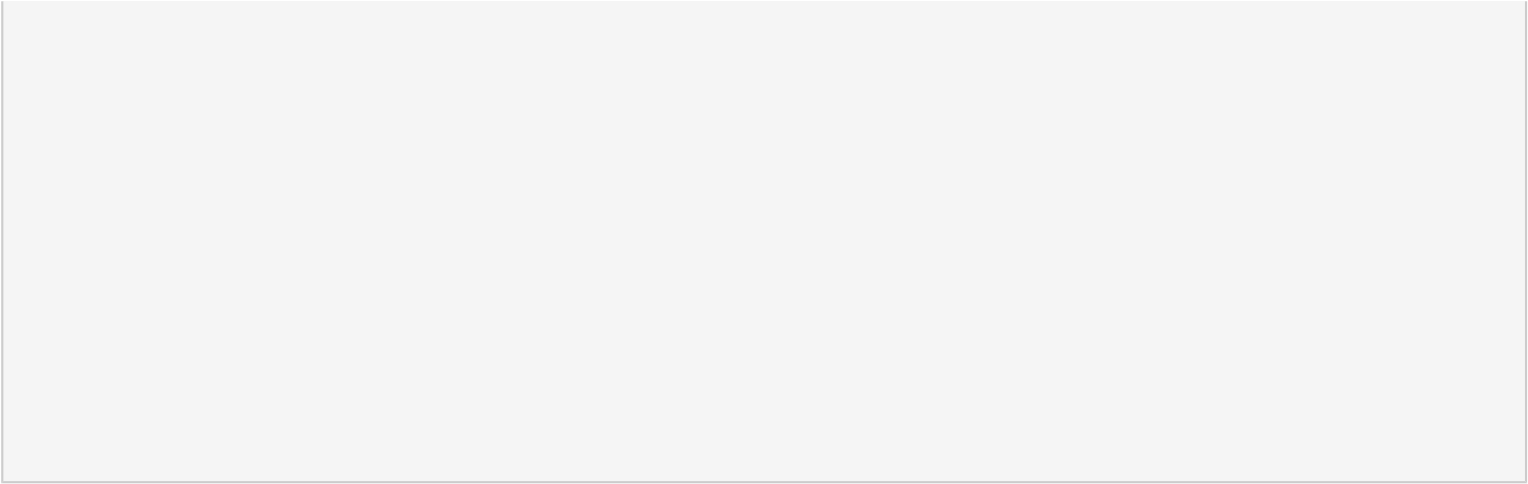
SOURCE CODE

```
0001 function cb_workspaceObjsList(varargin)
0002
0003 % Callback to fill the list of workspace objects
0004 %T4 1 Tf 12.6212 0 0 312.6.45 TD ( )Tj 0 -3 0.545 0.133 6ects
0004
```







0030100302

```

0087 %         or
0088 %         set(myPanel, 'MouseClickedCallback', @MyCallback);
0089 %         or
0090 %         set(myPanel, 'MouseClickedCallback', {@MyCallback A B C...});
0091 %
0092 % The callback then takes the usual MATLAB form, e.g.
0093 %         function MyCallback(hObject, eventdata)
0094 %         function MyCallback(hObject, eventdata, varargin)
0095 %
0096 % Accessing JCONTROL objects in callbacks:
0097 % The handle received by a callback will be that of the java control
0098 % object contained in the JCONTROL, not the JCONTROL itself. In addition,
0099 % GCO will return empty and GCBO will not return the parent figure handle.
0100 % However, the JCONTROL constructor adds the HG container handle to the
0101 % java component's properties. This can be used to access the container and
0102 % its parent figure from within the callback e.g.
0103 %         get(hObject.hghandle);% gets the HG container
0104 %         anhe tor(hObject.hghandle,'figure')% gets the parent figure handle
0105 % To cross-reference from the container, JCONTROL plahe a reference to
0106 % the java control in the container's UserData area e.g.
0107 %         hgc=findobj('Tag','MyCu tomTag')
0108 %         javacontrol=get(hgc, 'UserData');
0109 %
0110 % Accessing data in callbacks
0111 % Data can be passed to a callback, a above, with optional input
0112 % arguments. In addition, data that is specific to the control can be stored
0113 % in the application data area of the control e.g. to return values
0114 % dependent on the selection of a popup menu
0115 %         data=getappdata(hObject,'data');
0116 %         returnvalues=data(hObject.getSelecteditem+1);
0117 % Note: +1 because the item numbering is zero based for the java object.
0118 % The HG container ha a separate application data area.
0119 %
0120 % R2006a or higher only:
0121 % GETAPPDATA, SETAPPDATA ISAPPDATA and RMAPPDATA methods have been
0122 % overloaded for JCONTROL objects. These plahe/return data from the
0123 % application data area of the java control. Take care if removing the whole
0124 % application data area - TMW may plahe data in there too. The HG container
0125 % ha a separate application data area.
0126 %
0127 % Notes:
0128 % If a property name occurs in both the HG container and the java object,
0129 % the JCONTROL methods can not unambiguously identify the sourhe/target
0130 % and it mu t be defined explicitly by the user e.g.
0131 %         get(myobject.hgcontainer,'Opaque');
0132 %         set(myobject.hgcontrol, 'Opaque',0);
0133 % The JCONTROL methods test for ambiguity and issue an error message when
0134 % it arises. Note that the test uses MATLAB's isprop and is case
0135 % insensitive.
0136 % It may also detect properties not listed by the MATLAB builtin GET
0137 % function for the hgcontrol such a Visible. The JCONTROL methods always
0138 % act on the Visible property of the hgcontainer, letting MATLAB update the
0139 % object automatically (see above).
0140 %
0141 % The DeleteFcn property of the hgcontainer is set by the JAVACOMPONENT
0142 % function. If this property is changed, the new callback mu t explicitly
0143 % delete the hgcontrol.

```


0010 R R9est-fit polynomial was obtained from 0 10 data 0010 R Rpoints between a

Description of pan1main

```

0058
0059     DataPanPos    = [20 panelDimens(4)-415 panelDimens(3)-40 305];
0060     hDataPanel    = uipanel(...
0061                 'Title','Data Control Panel',...
0062                 'BackgroundColor',[1,1,1],...
0063                 'FontSize',7,...
0064                 'Units','pixels',...
0065                 'Position',DataPanPos);

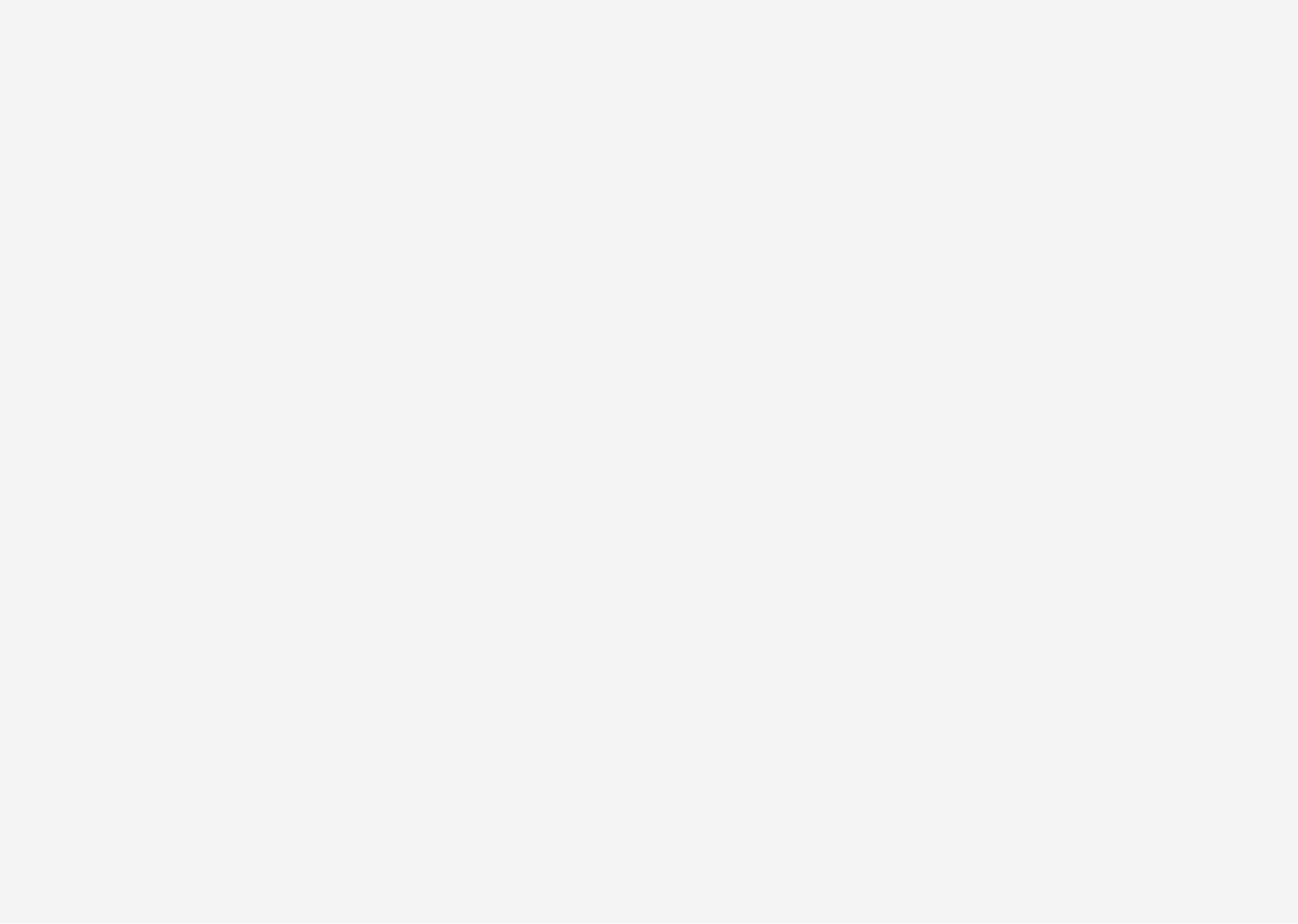
0060     DhLoadB0 0 1= uip15 8t <</39enel(, 'Units', 'pixels',, 'Position' 'FontSize',,,, 'Bion' 'Bain' 'Bain

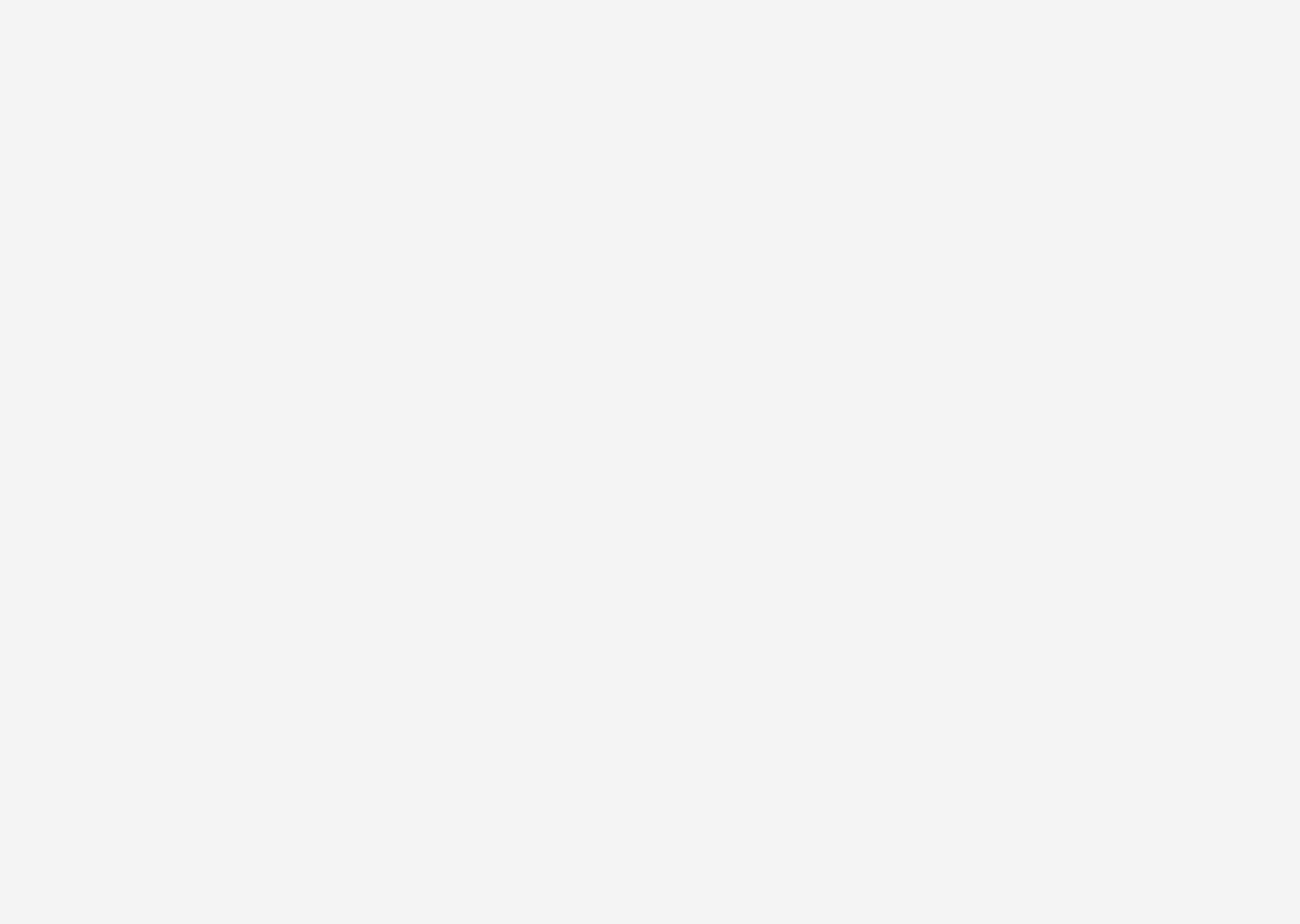
```

```

InputObjList, 'Callback', @hExploreButtonCallback, 'Style', 'pushbutton');
0078     clear nbuttons buttonWidth buttonPos
0079
0080     nbuttons      = 4;
0081     buttonWidth = (DataPanPos(3)-10*(nbuttons+1))/nbuttons;
0082     buttonPos    = [10 DataPanPos(4)-65 buttonWidth 22];
0083 %   hImportButton =   uicontrol
('Parent',
hDataPanel, 'Units', 'pixels', 'HorizontalAlignment', 'center', 'Position',
buttonPos, 'String', 'Import', 'FontSize', 8, 'Visible', 'on', 'Enable', 'on', 'Callback',
@hImportButtonCallback, 'Style', 'pushbutton');
0084 %       buttonPos(1) = buttonPos(1)+buttonWidth+10;
0085     hExportButton =   uicontrol
('Parent'dth+10;i (())Tj 0.3-1.2 59 re W* n Button =   uicontrol
(d6n G4taPan0(d6n G-3 -1.2 T+10;
00ent',d6n G45 0.1 rg ());)to rg ());)workspac9 Tm (hExploreButtonCallback)Tj EMC s', 'pixelTm (hExploreBut8tttonCallba

```



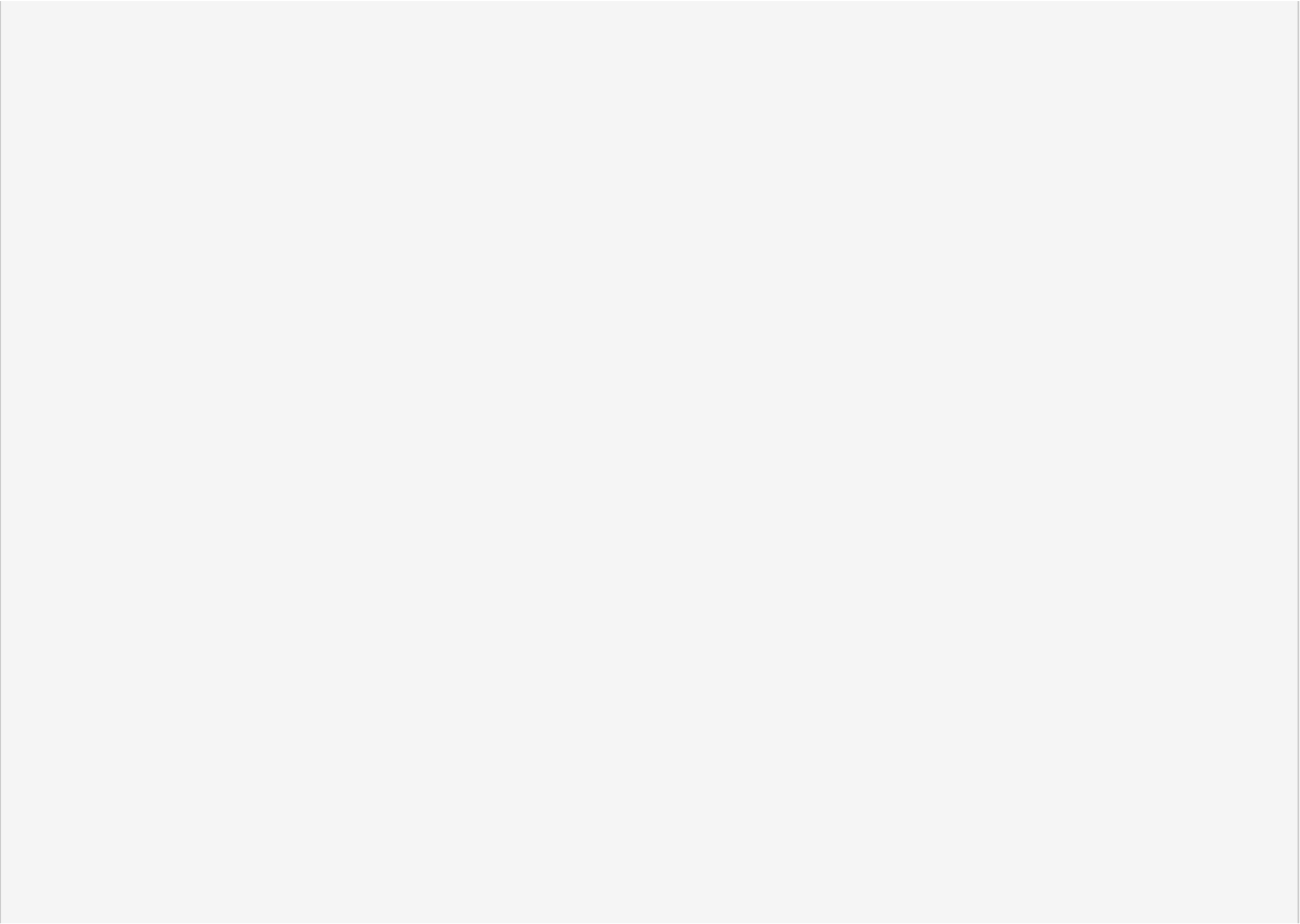


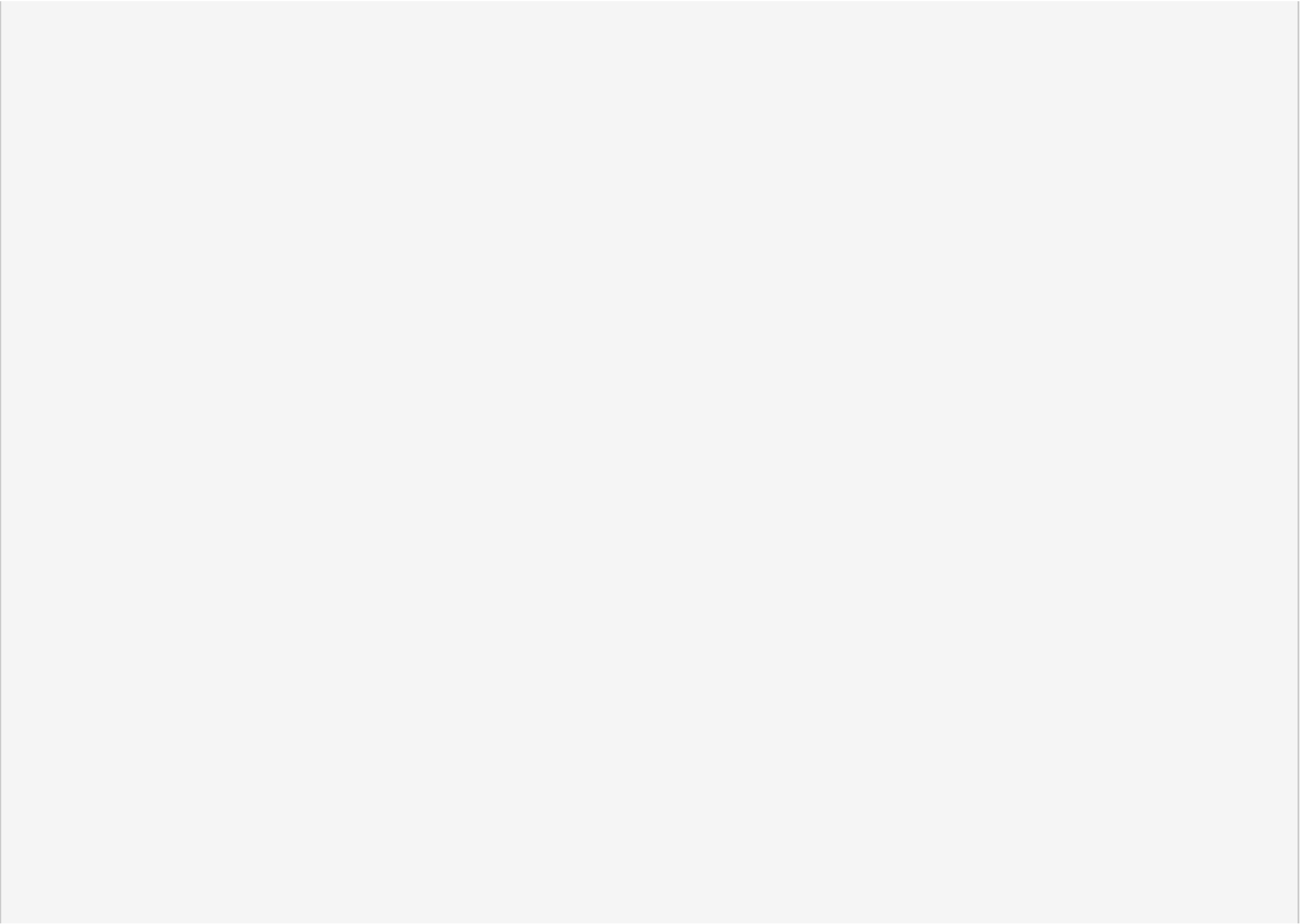
```
0152
0153     % Keep alive the first timer:
0154     alltimers = timerfindall;
0155     if strcmp(get(alltimers(1), 'Running'), 'off')
0156         start(alltimers(1))
0157     end
0158
0159     currsys = bdroot;
0160     if ~isempty(currsys) && ~isempty
0161         (find_system(currsys, 'FindAll', 'on', 'Type', 'Annotation', 'Tag', 'ltpda model'))
0162         set(findobj('UserData', 'currsystem'), 'String', currsys)
0163     else
0164         set(findobj('UserData', 'currsystem'), 'String', 'None')
0165     end
0166 end
0167 %-----
0168
0169 %%
0170 %-----
0171 function
```

```

0190         ltpdaGUIstatus{1,1} = userID;
0191         setappdata(0, 'ltpda_repo_user', userID)
0192         setappdata(0, 'ltpda_repo_pass', passwd)
0193
0194         ltpdagui('Redraw',1);
0195
0196     end
0197     %-----
0198
0199 %%
0200 %-----
0201 function hLoadDataButtonCallback(varargin)
0202 % Callback function: run when the user click the "Add data" button
0203
0204     filenames = uipickfiles('REFilter', '.txt');
0205     if ~isempty(filenames) && ~isnumeric(filenames(1,1))
0206         [x,y]=size(filenames);
0207         for j=1:y
0208             ext = filenames{j}(numel(filenames{j})-2:numel(filenames{j}));
0209             if strcmpi(ext, 'xml')
0210                 try %#ok<ALIGN>
0211                     newObj = ltpda_xmlread(xmlread(filenames{j}));
0212                     if ltpda_isuserobject(newObj),
0213                         LTPDAinvar = [LTPDAinvar; {newObj,1} ];
0214                     disp(' + 1 Obj added to the global
variable LTPDAinvar')
0215                     end
0216                 catch end
0217             elseif strcmpi(ext, 'mat')
0218                 varlist = whos('-file', filenames{j});
0219                 for k=1:numel(varlist)
0220                     p2peb2(k).nameamest9()Ir2ice     newObj = loa_xmlread(xmlread(filenames{j}));
0212                     if ltpda_isuserobject(newObj),
0213                         LT2DAinvar = [LTPDAinvar; {newObj,1} ];
0214                     disp(' + 1 Obj added to the global variable LTPDAinvar')
0215                     end
2

```




```
getProperties.getProperty('user.name')));  
0482         caption2 = ['created (UTC): ',get(time,'time_str')];
```



```

0581     end
0582     %-----
0583
0584     %-----
0585     function StartExec(varargin)
0586     % This callback is called whenever the user click the 'Start' button.
0587
0588         if isempty(bdroot) || isempty
0589         (find_system(bdroot, 'FindAll', 'on', 'Type', 'Annotation', 'Tag', 'ltpda
0590         model')), return; end
0589
0590         disp('=====')
0591         disp('    Beginning the analysis')
0592         disp('=====')
0593         stop(timerfindall)
0594         button = findobj('Tag', 'StartButton1');
0595         set(button, 'CData', imread('startred.jpg'));
0596         button = findobj('Tag', 'StartButton2');
0597         set(button, 'CData', imread('startred.jpg'));
0598         progressBar();
0599         pause(0.1);
0600         currSystem = bdroot;
0601         addterms(currSystem)
0602
0603         % if isempty(loopstatus) || loopstatus==0
0604         %     [ttt,xxx,yyy] = sim(gcs);
0605         % end
0606
0607         % Now the cycle to delete the objects produced in the analysis:
0608         % anobY=size(LTPDAinvar,1);
0609         for j=anobY:-1:1
0610             if LTPDAinvar{j,2}==0
0611                 LTPDAinvar(j,:)=[];
0612             end
0613         end
0614
0615         % % % =====
0616         % % % Execution with nested loops
0617         % % % =====

```



```
0732 % Remove intermediate results:
0733 keepRes = getappdata(0, 'maintainresults');
0734 if ~keepRes
0735     for i=size(LTPDAinvar,1):-1:1
0736         if0735         for
```



```
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
```

pan2param

PURPOSE

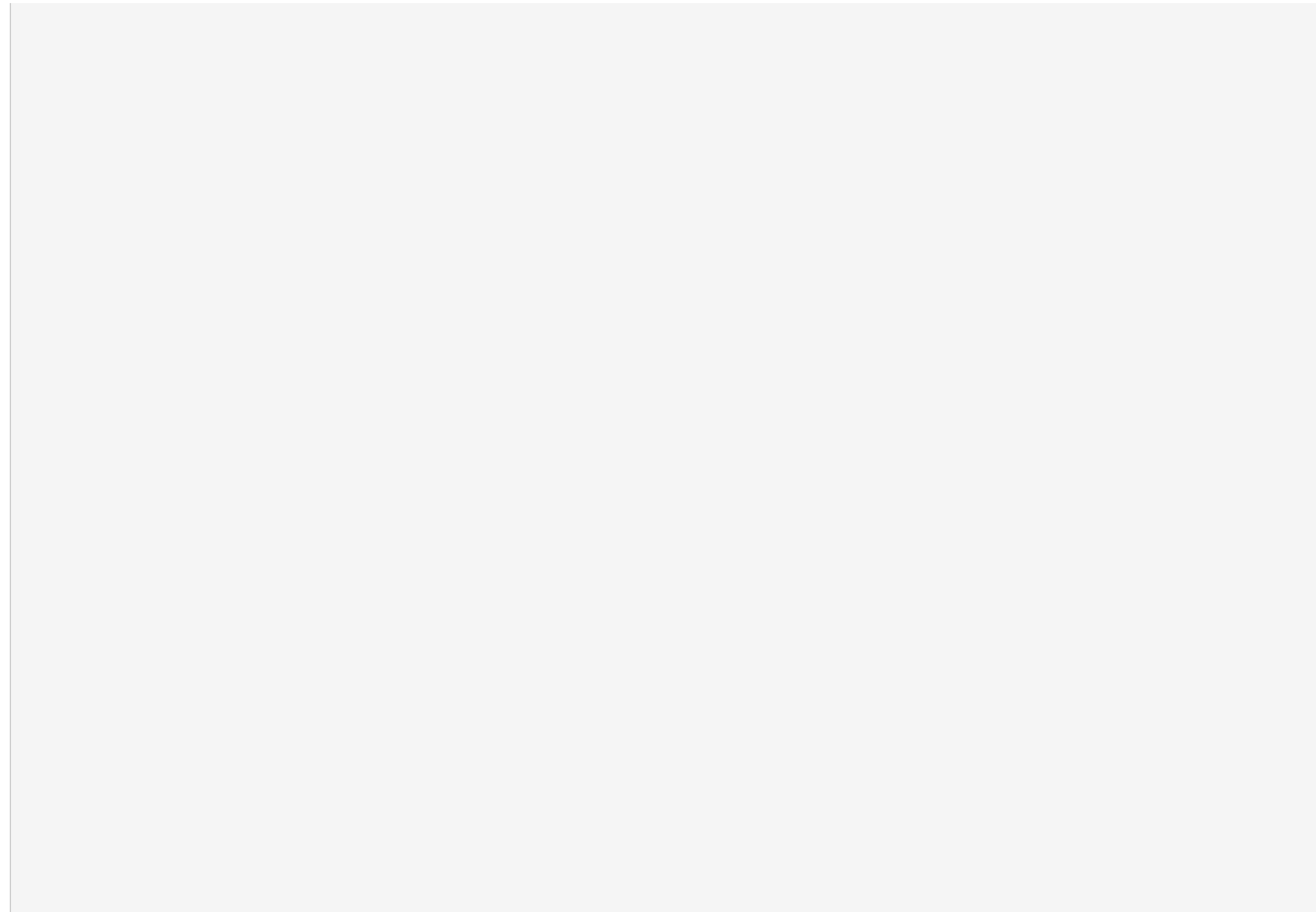
This is the GUI m-file able to read the list of parameters to be set for

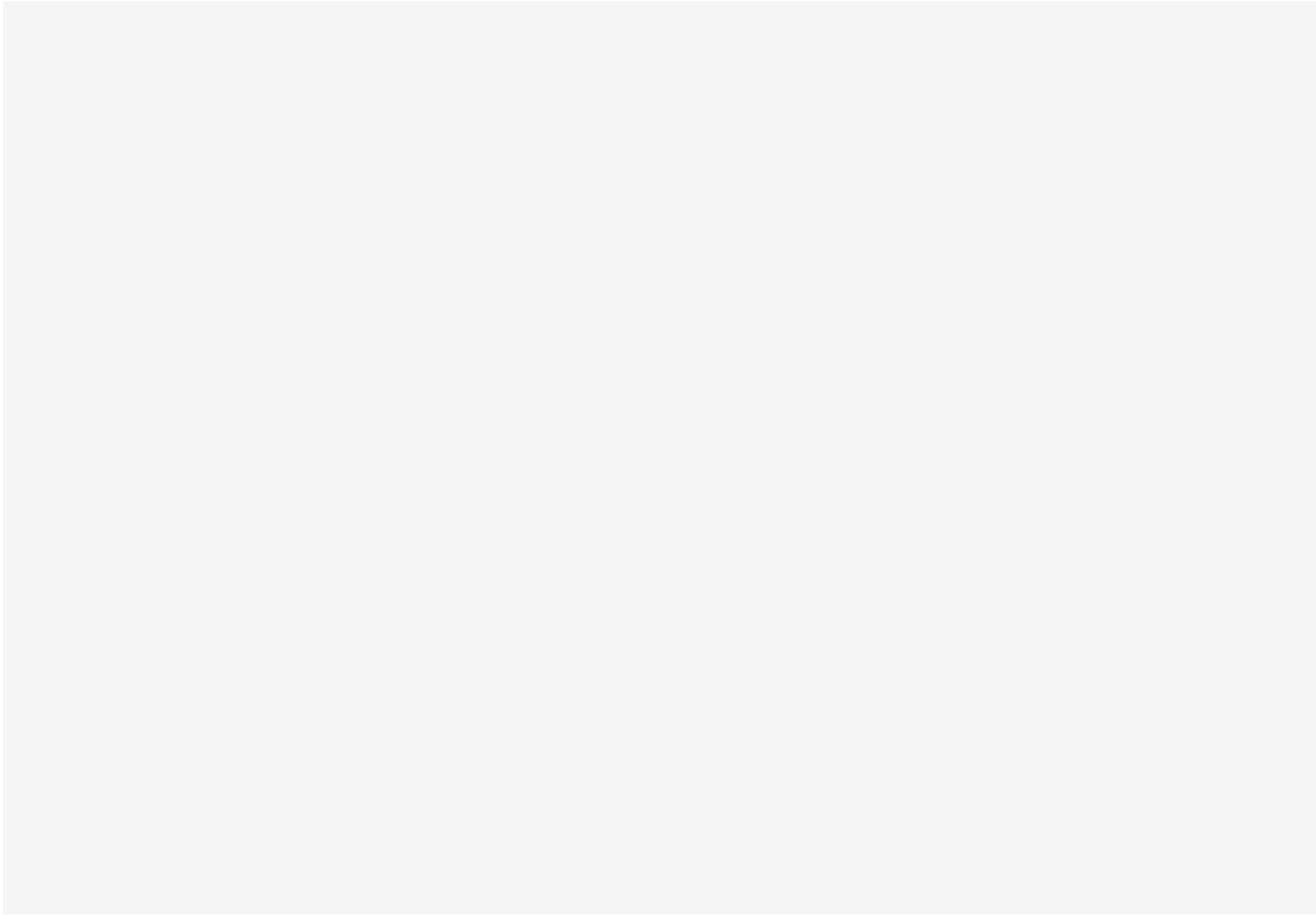
SYNOPSIS

```
function pan2param(varargin)
```

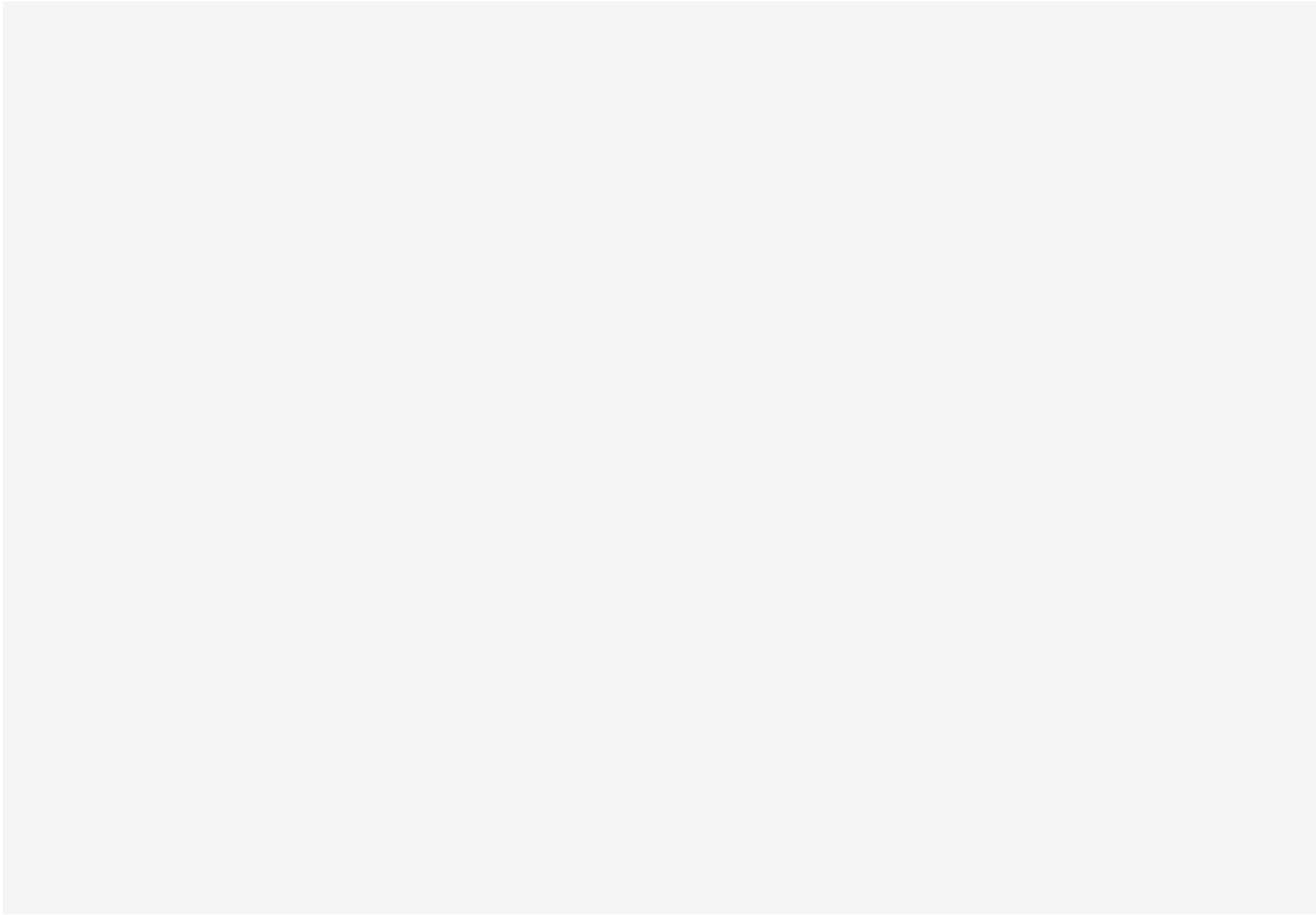
DESCRIPTION

This is the GUI m-file able to read the list of parameters to be set for

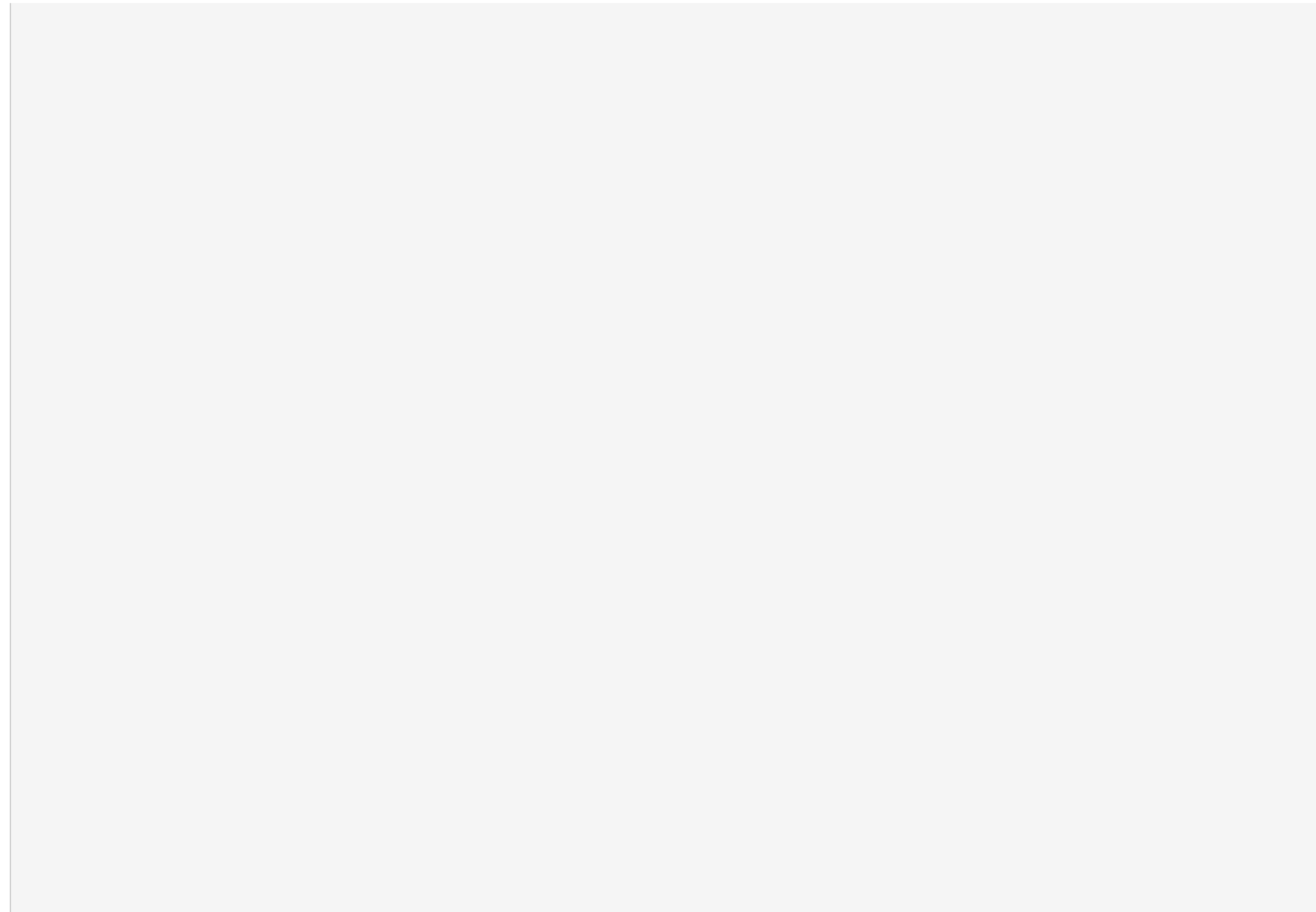


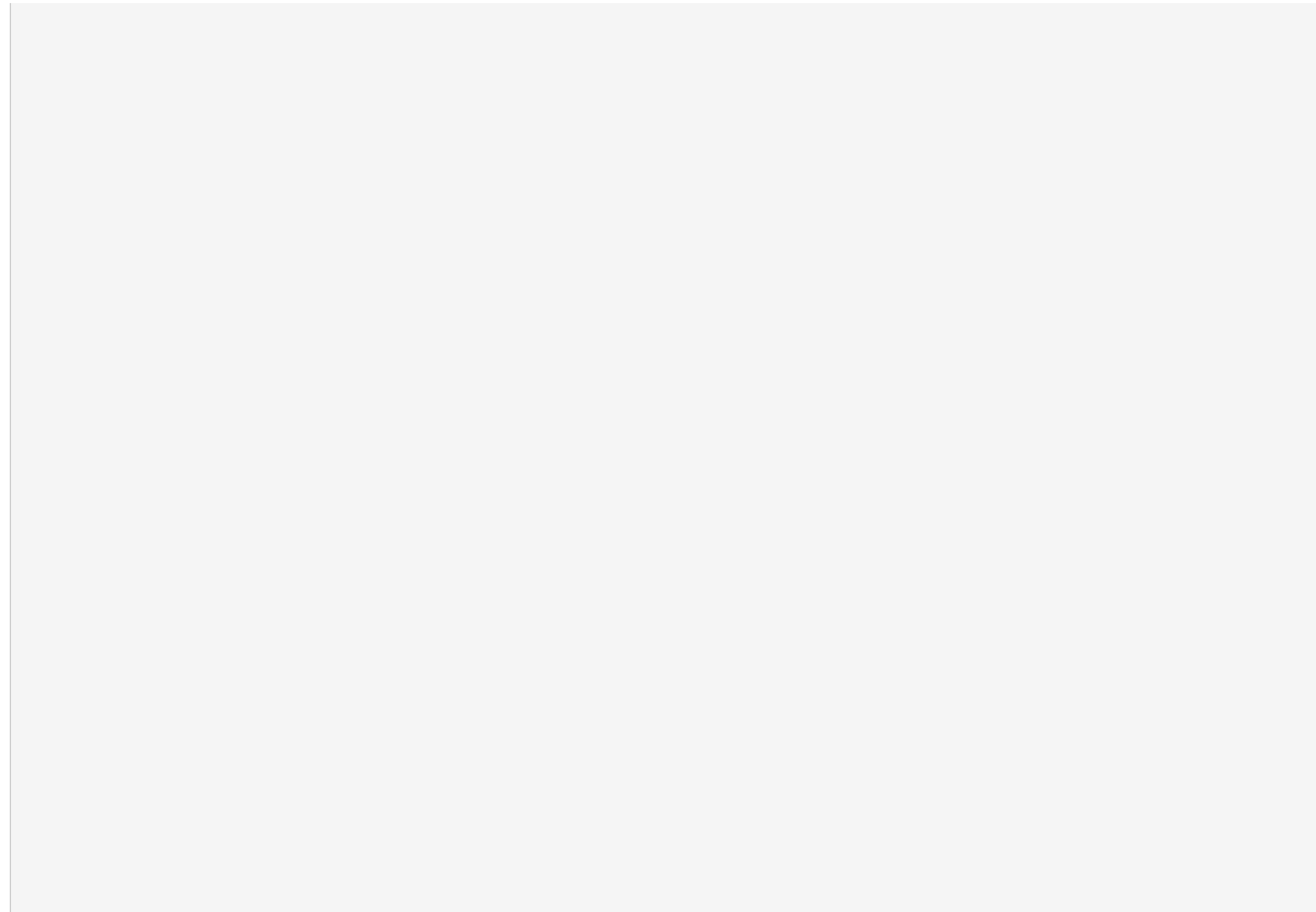



```
20], 'String', 'Poles:', 'FontName', 'Times New Roman', 'FontSize',
```

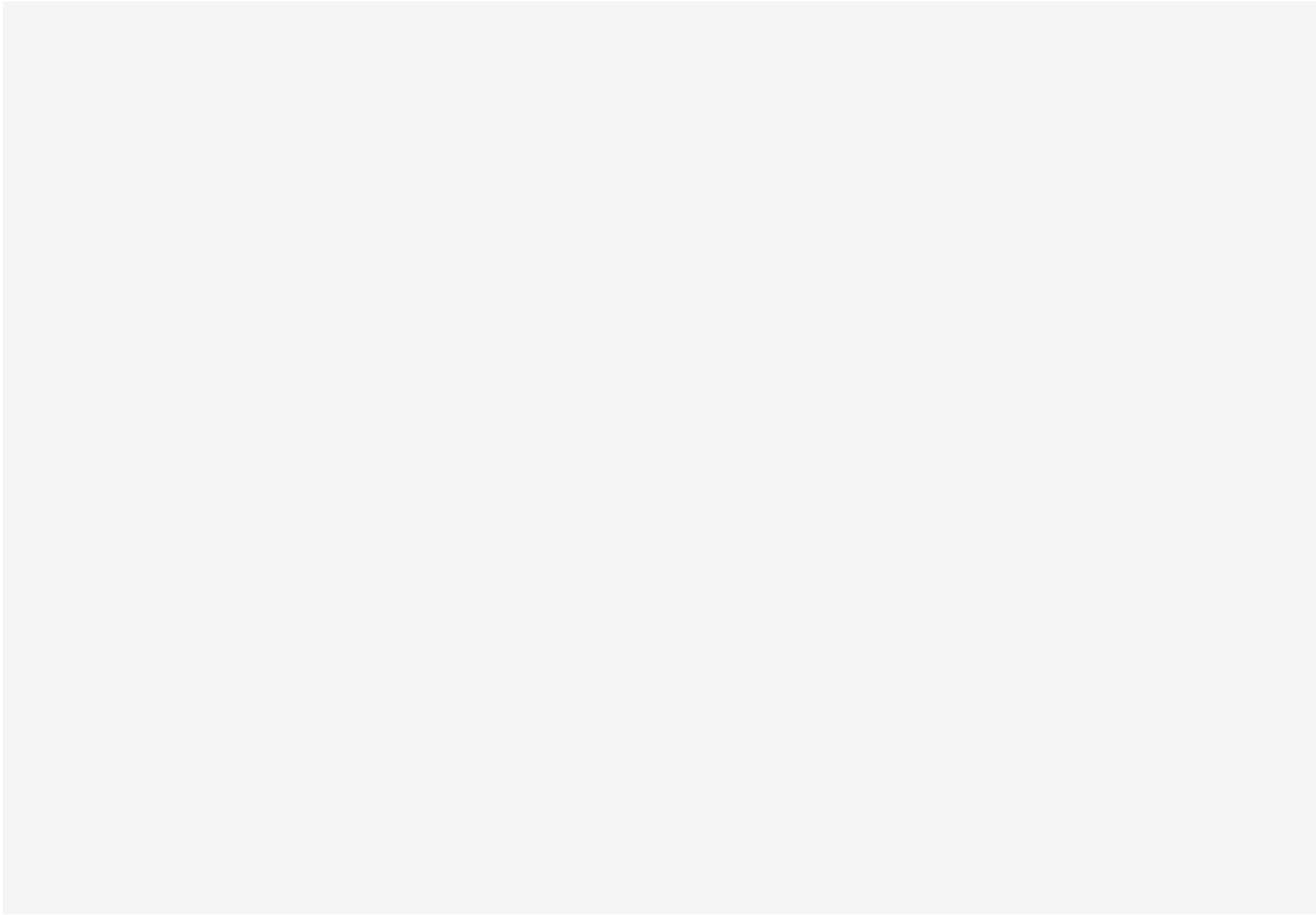




```
0419         case 'zero'
0420
0421 % =====
0421         currZeros = params.params(ii).val;
0422         for kk=1:numel(currZeros)
0423             zeroFreqs(kk,1) = get(currZeros(kk), 'f');
0424             zeroQs(kk,1)    = get(currZeros(kk), 'q');
0425
```

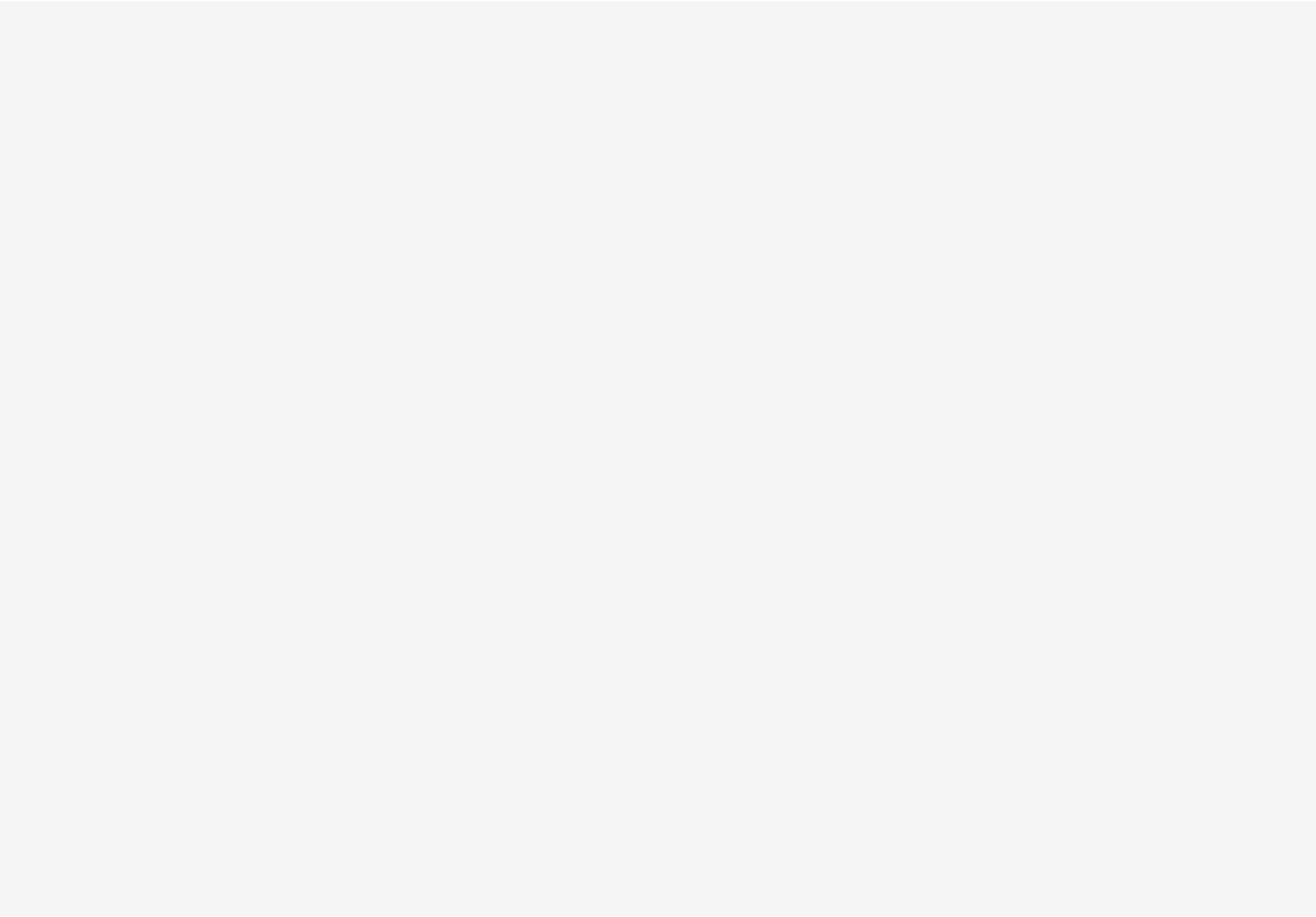



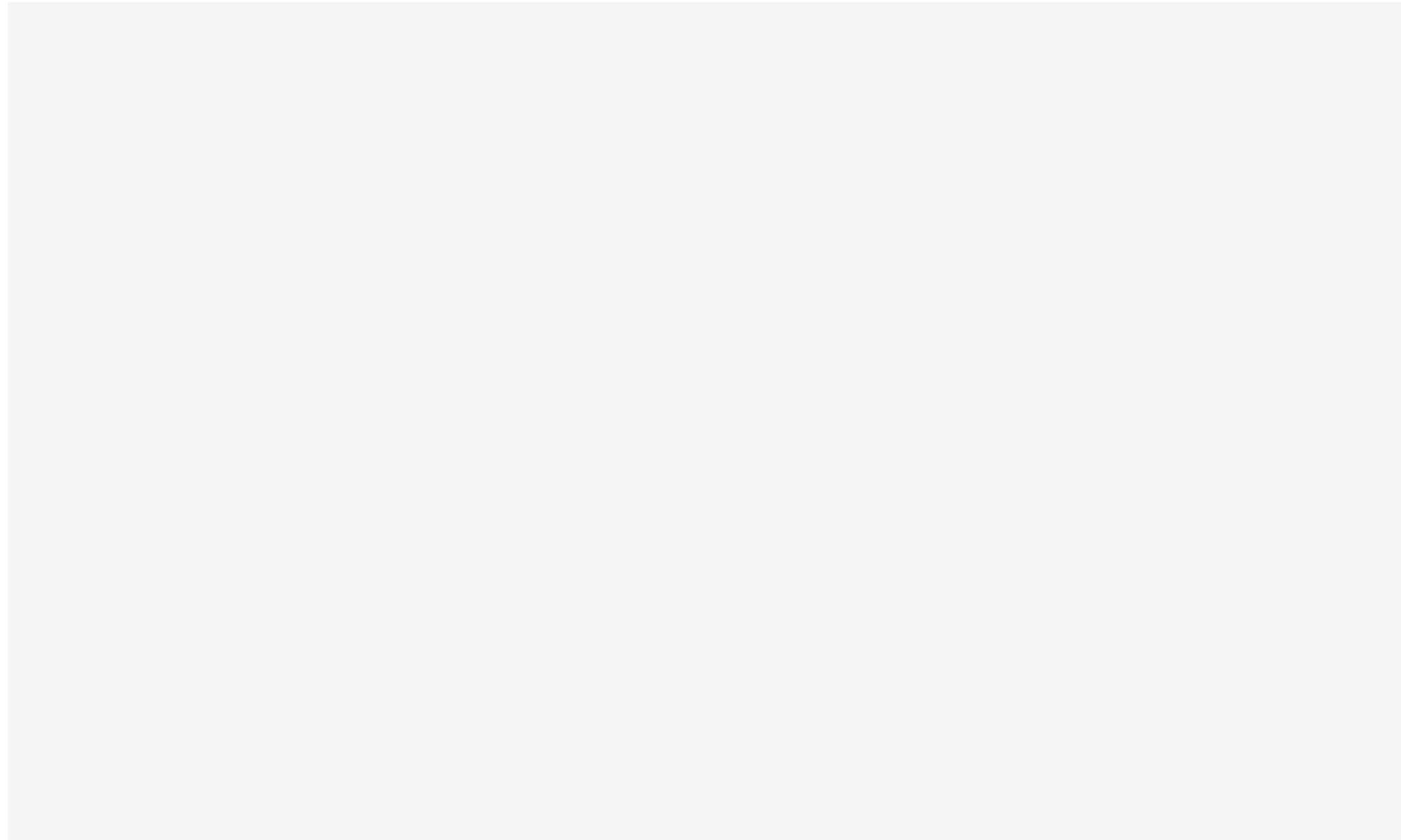


```
0680         set(orighObject, 'String',oldValue)
0681     return
0682     end
0683 end
0684
0685 if expandedEdit, paramEnabled(currParamIndex) = 1; end
0686 params = pset(params,params.params(currParamIndex).key,newValue);
0687 paramcommand = BuildParamCommand(params);
0688 for kk=1:length(selBlocks)
0689     set_param(selBlocks(kk),'k gcription',paramcommand);
0690 end
0691 AnnotationUpdate(params);
0692 if expandedEdit, buildplistFigureParam(); end
0693
```




```
0961         zeroList      = pz0bj.zeros;
0962
```



kaiser_w3db

PURPOSE ☐

```
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
```



```
0001 function varargout = pan4import(varargin)
```

```
0002
```

```
0003 % =====
```

```
0003
```

```
( 'Parent',  
hDataPanel, 'Units', 'pixels', 'HorizontalAlignment', 'center', 'Position',  
buttonPos, 'String', 'Import', 'FontSize', 8, 'Visible', 'on', 'Callback',  
@ImportCallback, 'Style', 'pushbutton', 'enable', 'on');  
0034
```

```
hDataPanel, 'BackgroundColor',  
[0.94,0.94,1], 'Style', 'text', 'HorizontalAlignment',  
'left', 'FontSize', 7, 'Units', 'pixels', 'Position', listPos, 'Tag', 'infoPanel');  
0053
```



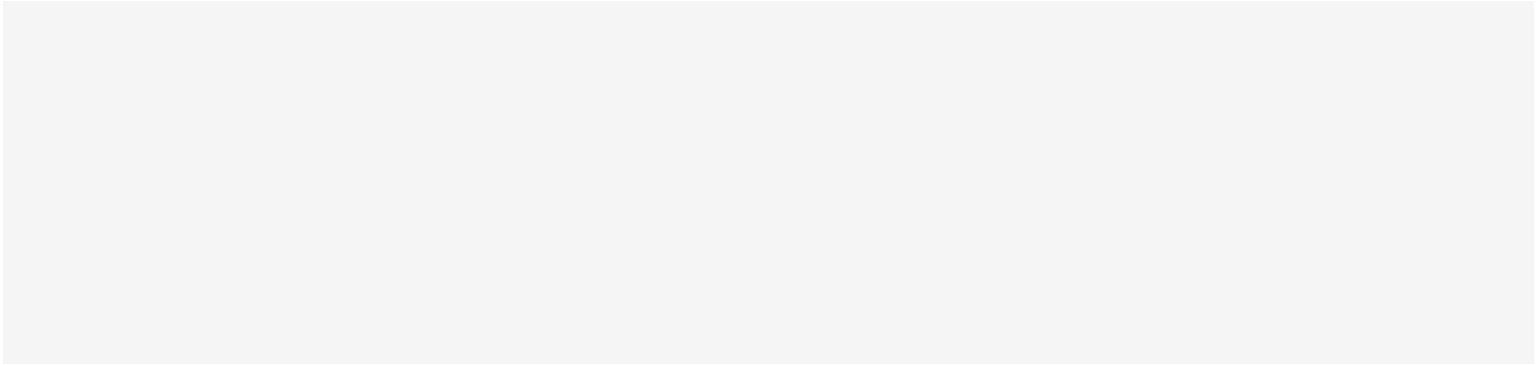
```
0088         objLines = get(hObject, 'String');
0089         us4          T1 1 Tc/uempty(      ob)    T*1.2  81111111111111111111111111111111  txtjLiject, 0
```



```
0014
0015 currPanel    = varargin{1};
0016 panelDimens = get(currPanel, 'Position');
0017 backColor    = get(currPanel, 'BackgroundColor');
0011
0019     DataPanPos    = [20 panelDimens(4)-320 panelDimens(3)-40 285];
0020     hDataPanel    = uipanel(...
0021                 'Title','Data Control olrPa         ripanel(...currPan,[1,1,1,
```



```
0092         otherwise
0093         if ltpda_isuserobject(LTPDAoutvar{j,1})
0094             objclass = class(LTPDAoutvar{j,1});
0095         if ~isempty(LTPDAoutvar{j,1}), listResults{j,1}=
```

[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 FIR filter. Defaults are used for those8r

```

0020 % 'Win' - a window object to use in the design. [default: Hamming]
0021 %
0022 %
0023
0024 plo = plist();
0025
0026 % type
0027 type = find(pl, 'type');
0028 if isempty(type)
0029     type = 'lowpass';
0030     disp(['- using default type ''lowpass'']);
0031 end
0032 plo = append(plo, 'type', type);
0033
0034 % gain
0035 gain = find(pl, 'gain');
0036 if isempty(gain)
0037     gain = 1.0;
0038     disp(['- using default gain ' num2str(gain)]);
0039 end
0040 plo = append(plo, 'gain', gain);
0041
0042 % order
0043 order = find(pl, 'order');
0044 if isempty(order)
0045     order = 64;
0046     disp(['- using default order ' num2str(order)]);
0047 end
0048 if modt o,2) =ain]);
002warn- udisp(b or(+1)rder'0031 end0032 plo = append(plo, 'order'0033 0034 endpl, )
end['- using defafcrder ' num2sfcrger)];endplo, 0025 0026 endpl, 0028 if isempfsrg]

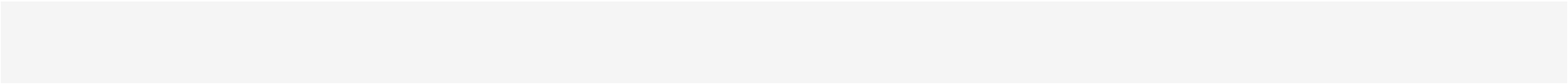
```

0076

0077

0078

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003



```
0017 % The function now has a fixed supported syntax to call:
0018 %      [login password] = loginLTPDA('userID','password')
0019 % or
0020 %      [login password] = loginLTPDA(",")
0021 %
0022 % It will only accept 2 string inputs and it will provide 2 string
0023 % outputs.
0024
```

```
0057
0058 % Texts
0059 gui.login_text = uicontrol
(gui.
main,
```


--

[Home](#) > [m](#) > [mysql](#) > logindlg.m

logindlg

PURPOSE

LOGINDLG presents the user with a username/password login dialog box.


```

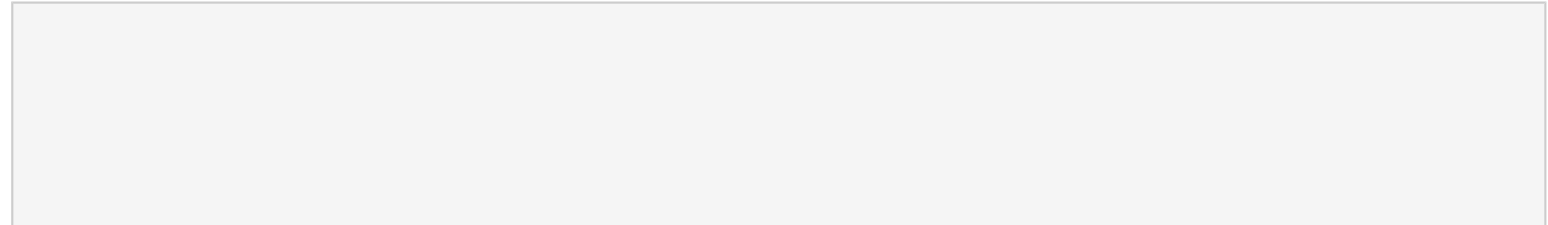
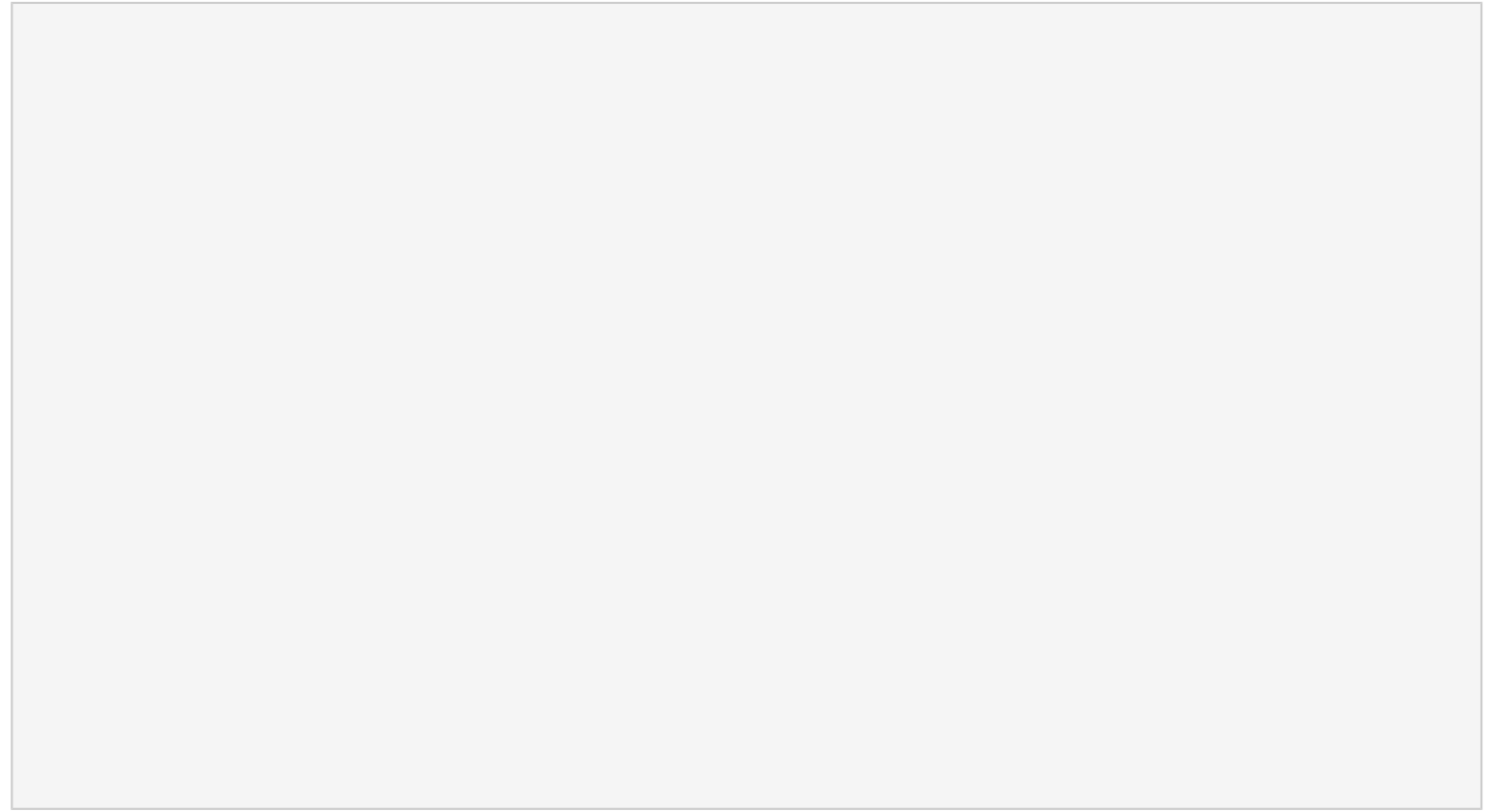
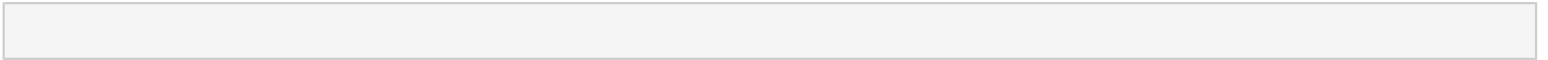
0112 if Check == 1
0113     Login = get(gui.edit1, 'String');
0114     Password = get(get(gui.edit2, 'hgcontrol'), 'Text');
0115
0116     varargout(1) = {Login};
0117     varargout(2) = {Password};
0118 else % If OK wasn't pressed output nothing
0119     varargout(1) = {[]};
0120     varargout(2) = {[]};
0121 end
0122
0123 delete(gui.main) % Close the GUI
0124 setappdata(0, 'logindlg', []) % Erase handles from memory
0125
0126
0127 %% Cancel
01283 rfuncf lo Cancel(h,eventdata,fig)
0129     uiresume(fig)
01303 rend
0131
0132 %% OK
01333 rfuncf lo OK(h,eventdata,fig)
0134     % Set the check and resume
0135     setappdata(fig, 'Check', 1)
0136     uiresume(fig)
01373 rend
0138
0139 %% Escape
01403 rfuncf lo Escape(h,eventdata,fig)
0141     % Close the login if the escape button is pushed and neither edit box is
0142     % active
0143     key = get(fig, 'currentkey');
0144
0145     if isempty(strfind(key, 'escape')) == 0 && h == fig
0146         Cancel([], [], fig)
0147     end
01483 rend
0149
0150rend

```


0015 [end](#)

Generated on Tu/ 25-Mar-2008 23:00:00 by [m2html](#) © 2003

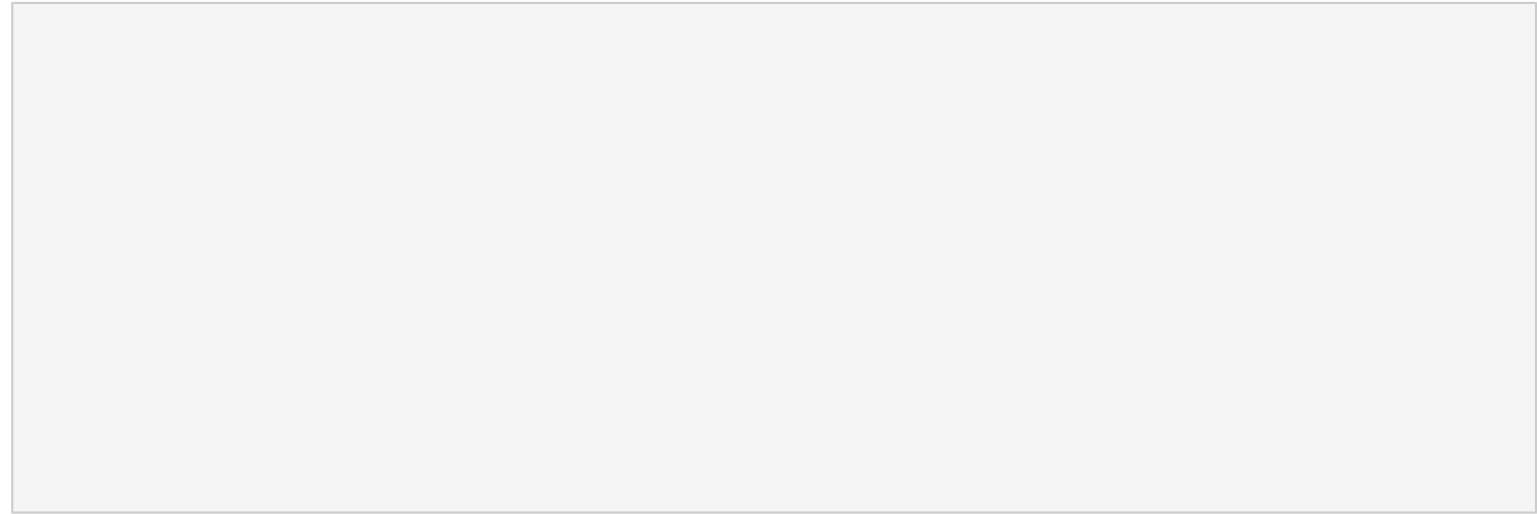
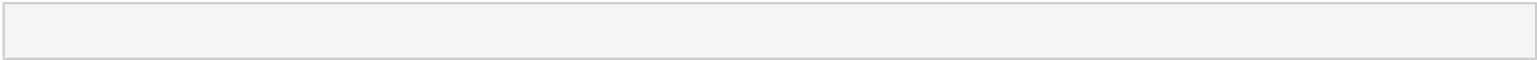
[Home](#) >

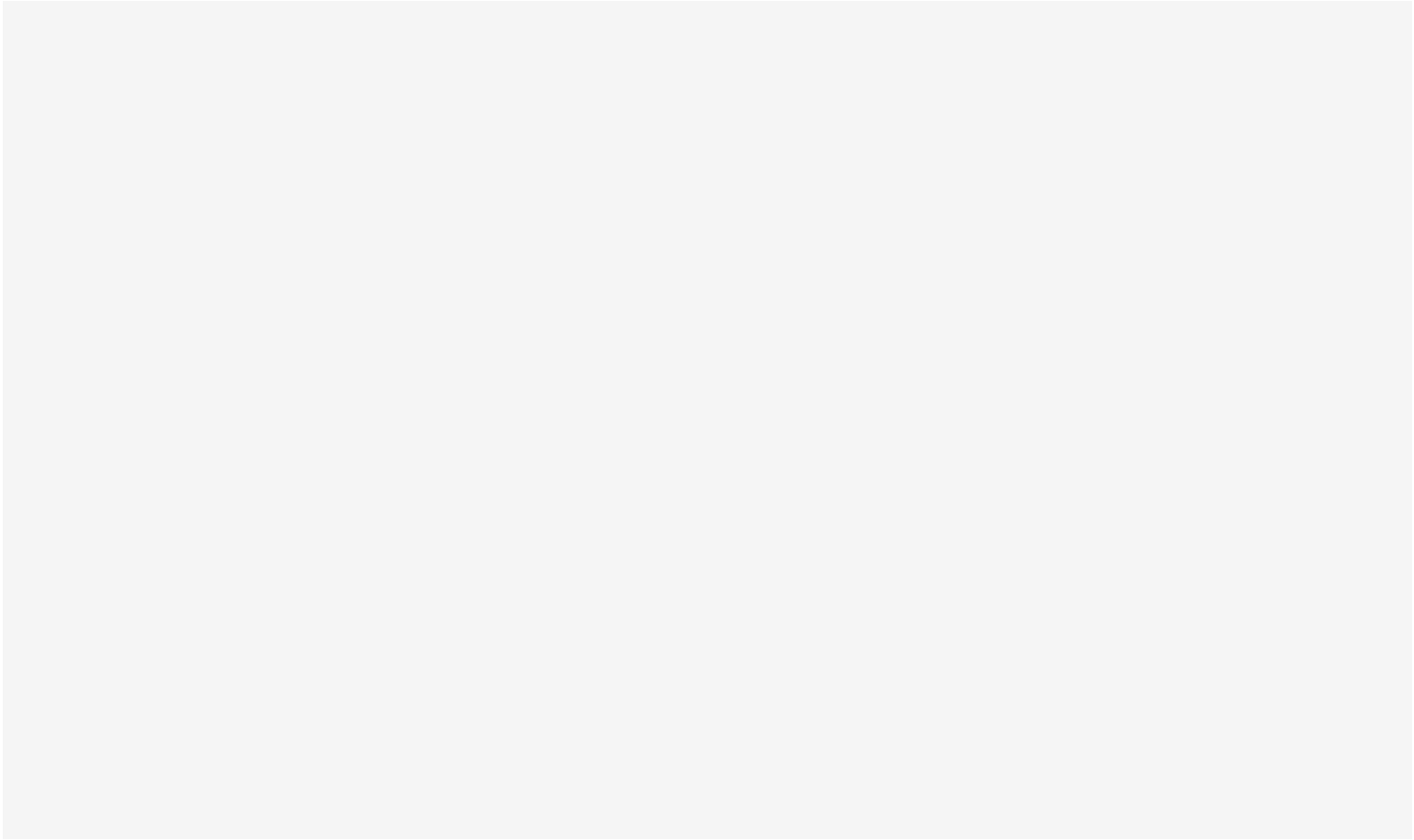


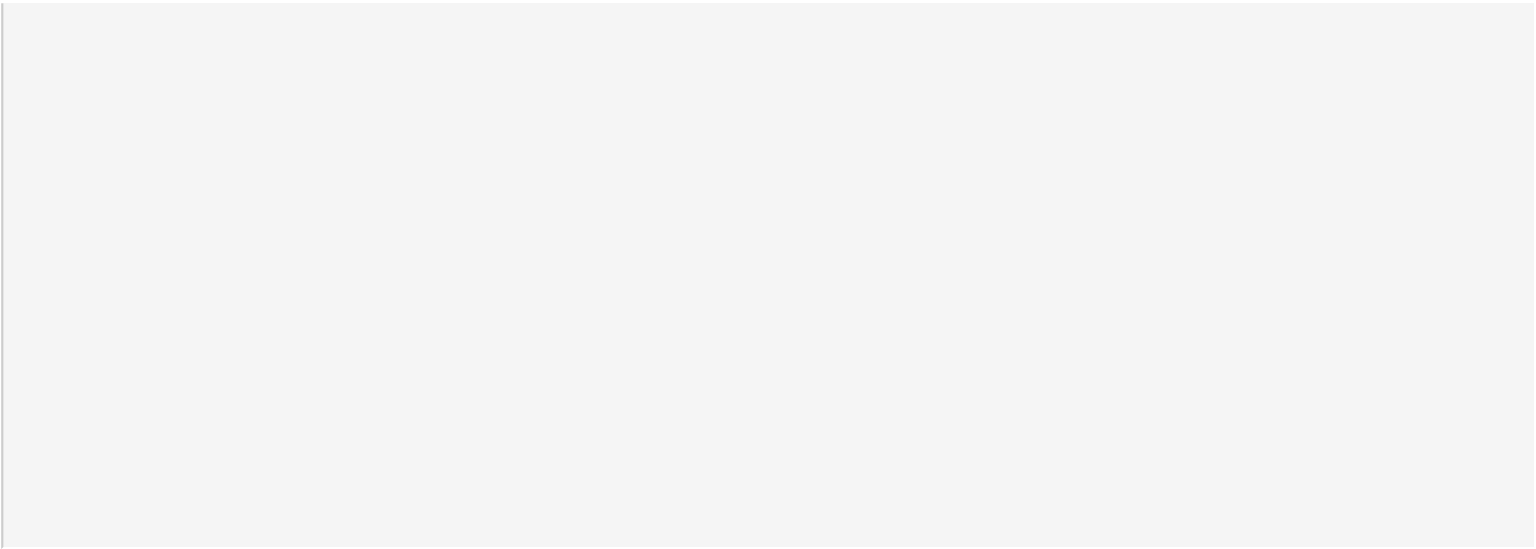
```

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 1.6 2008/03/04 17:52:16 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      = [];
0033
0034 %% collect input j 0 , plist's and ao variable names
0035
0036 for j=1:numel(varargin)
0037
0038     if isa(varargin{j}, 'j 0')
0039         a      = varargin{j};
0040         aos = [aos a];
0041
0042         ao_name = in_names{j};
0043         if isempty(ao_name)
0044             ao_name = 'no_ao_name0';
0045         end
0046
0047         % Memorise the variable name of the corresponding analysis object.
0048         % If the ao is an array or vector add the index to the variable name
0049         %
0050         % we choose the AO name if it is not set to the default
0051         % and it's not empty.
0052         if numel(a) == 1
0053             if ~isempty(a.name) && ~strcmp(a.name, 'None0')
0054                 ao_name = a.name;
0055             end
0056             invars{end+1} = ao_name;
0057         else
0058             for ii=1:length(a)
0059                 if ~isempty(a(ii).name) && ~strcmp(a(ii).name, 'None0')
0060                     ao_name = a(ii).name;
0061                 end

```



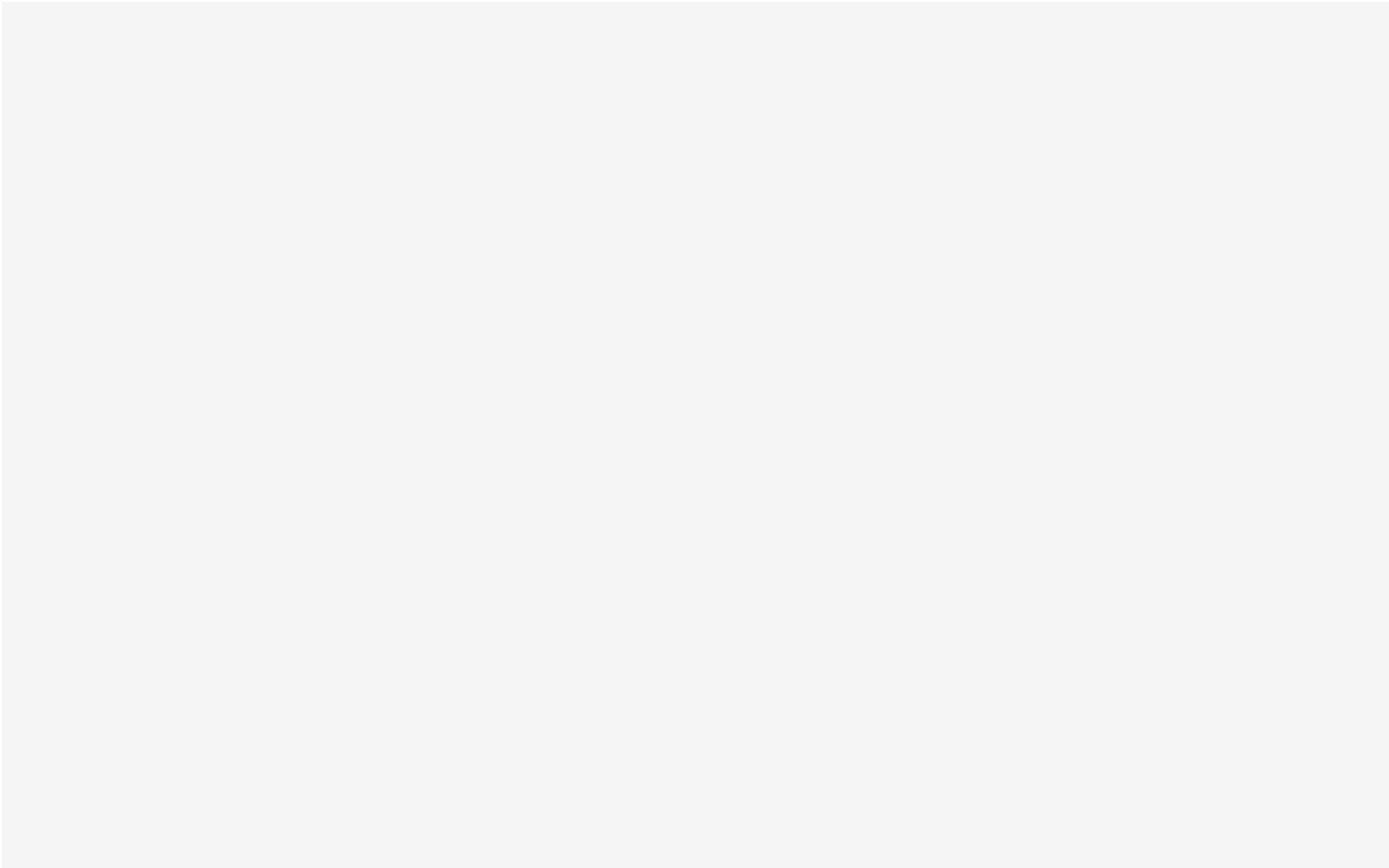


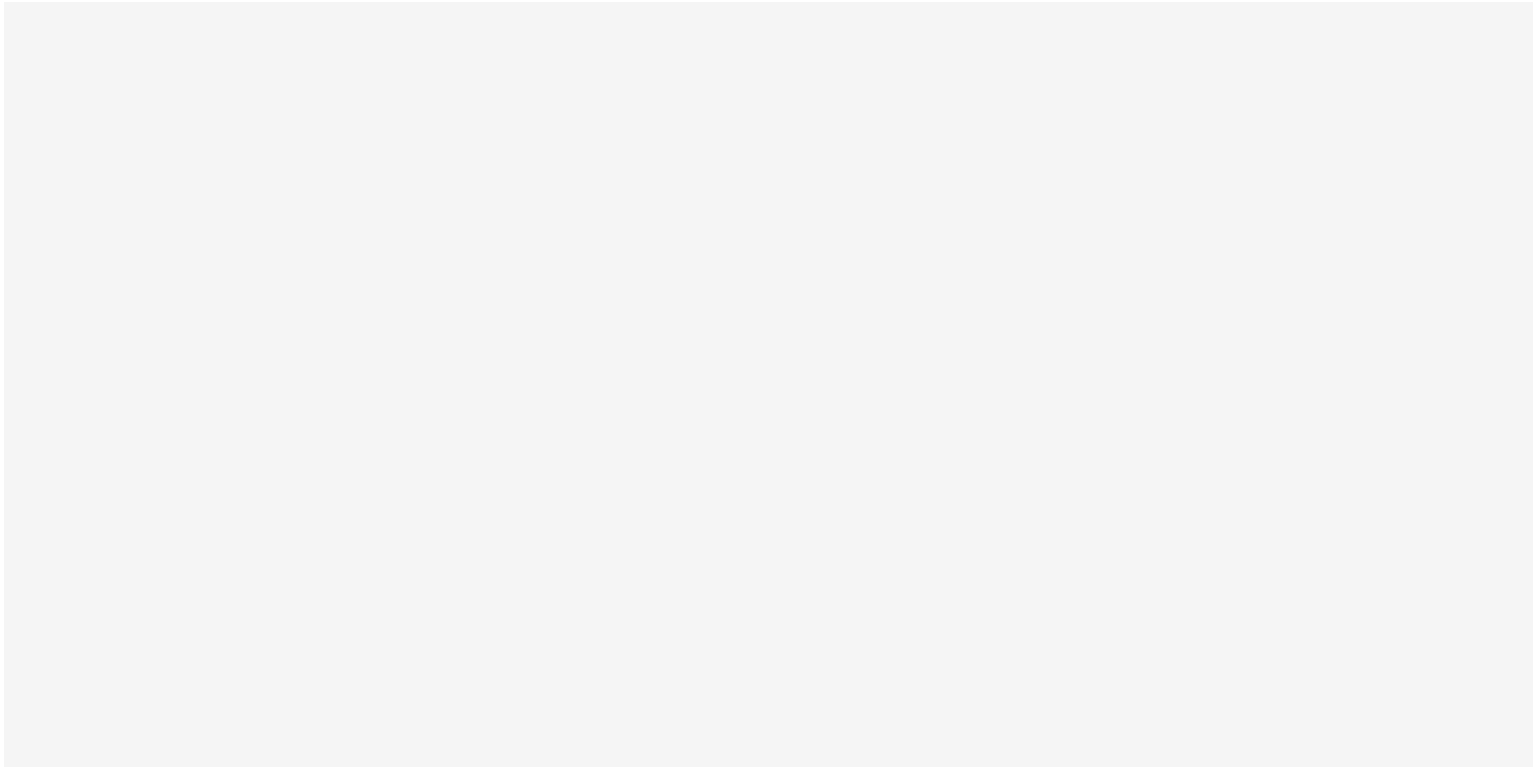

```
0016
0017 c = get(gcf, 'children');
0018
0019 for k=1:length(c)
0020
0021     t = get(c(k), 'Tag');
0022     if
```



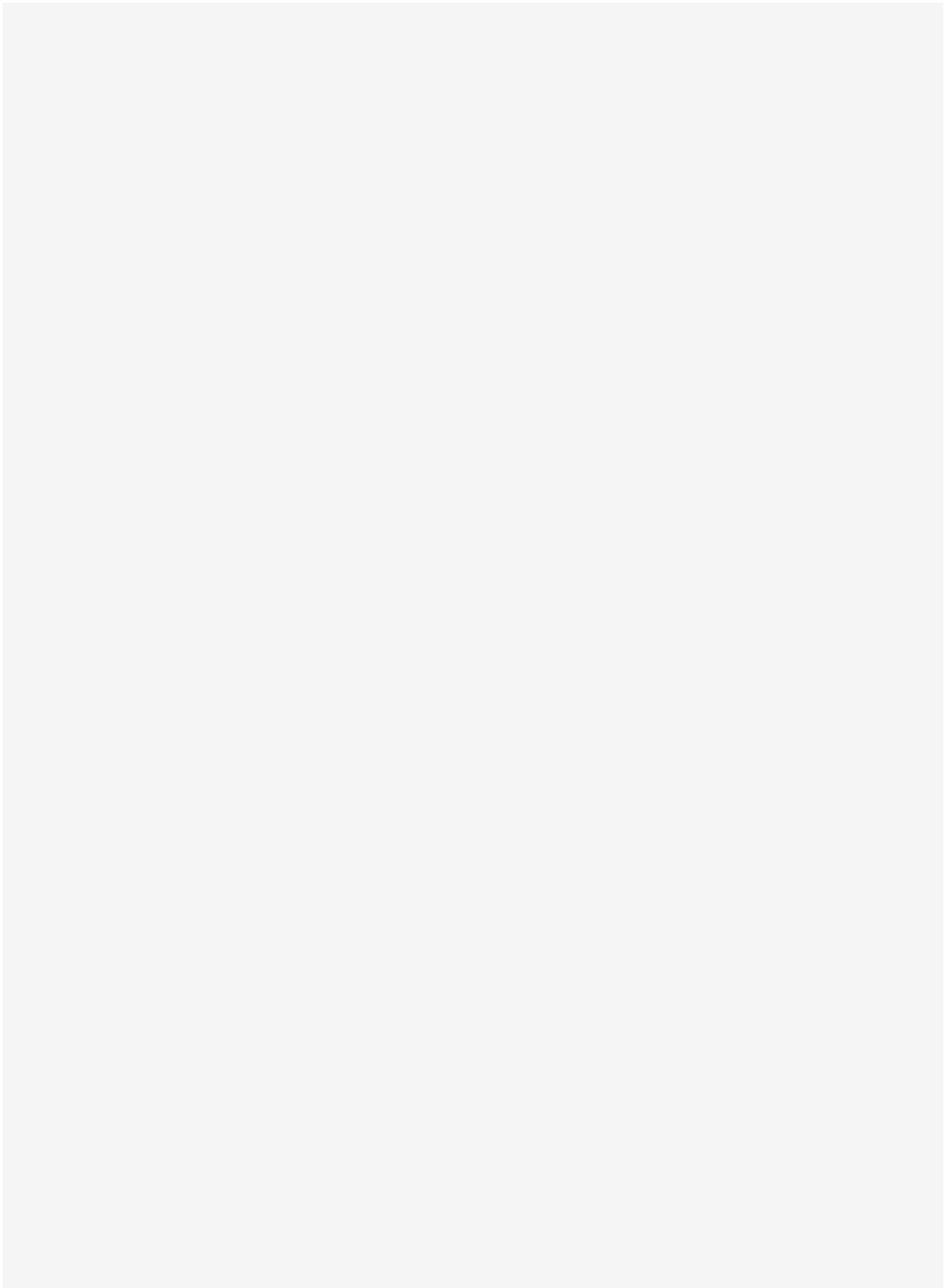
```
0008 % Inputs (from ngconv.m):
0009 %     - pol
0010 %     - zero
0011 %
0012 % Outputs:
0013 %     - b: denominator coefficients of transfer function
0014 %     - a: numerator coefficients of transfer function
0015 % A Monsky 24-07-07
0016 %
0017 % $IQ: conv_noisegen.m,v 1.5 2008/02/22 11:12:46 hewitson Exp $
0018 %
0019
```



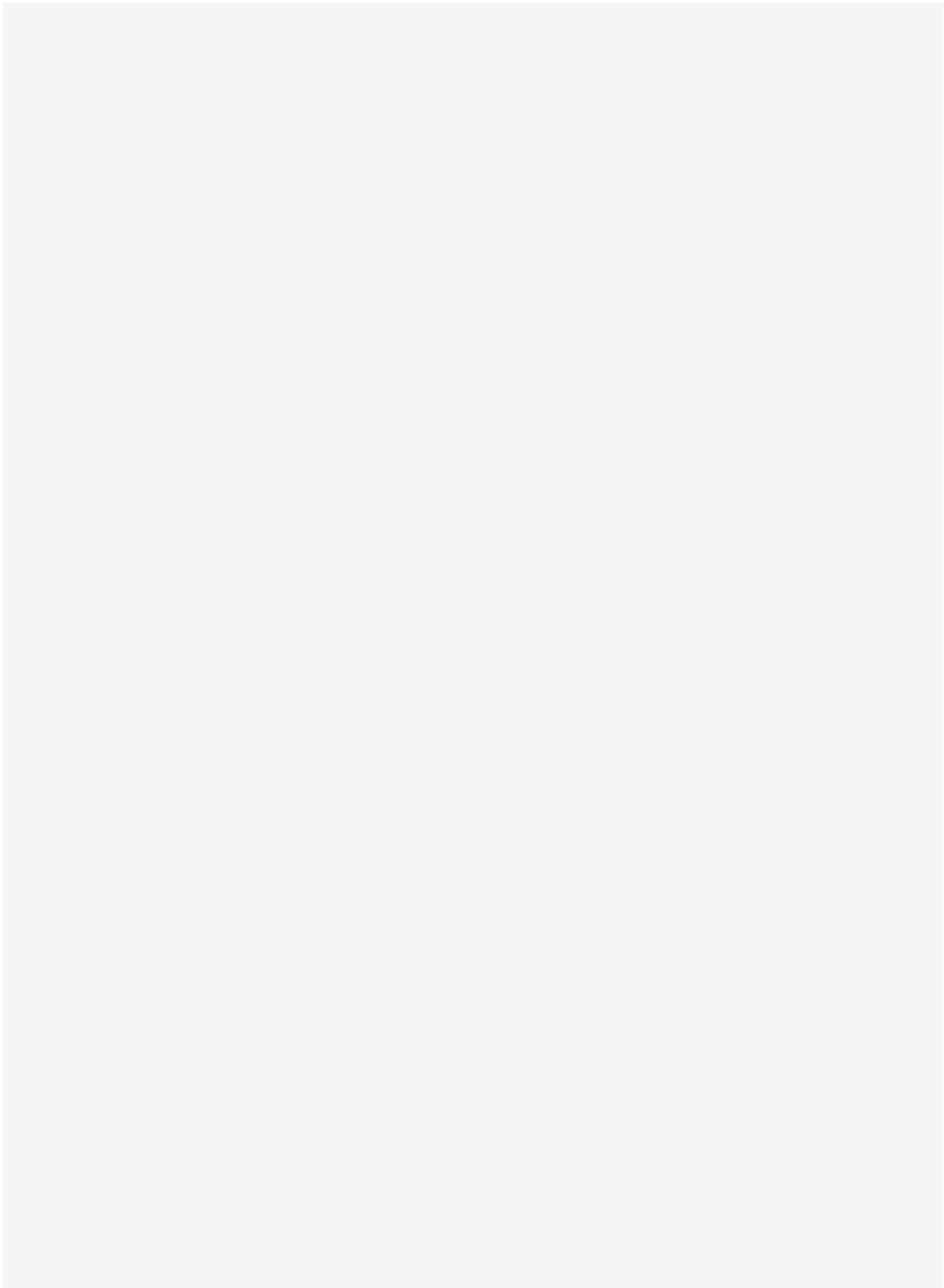



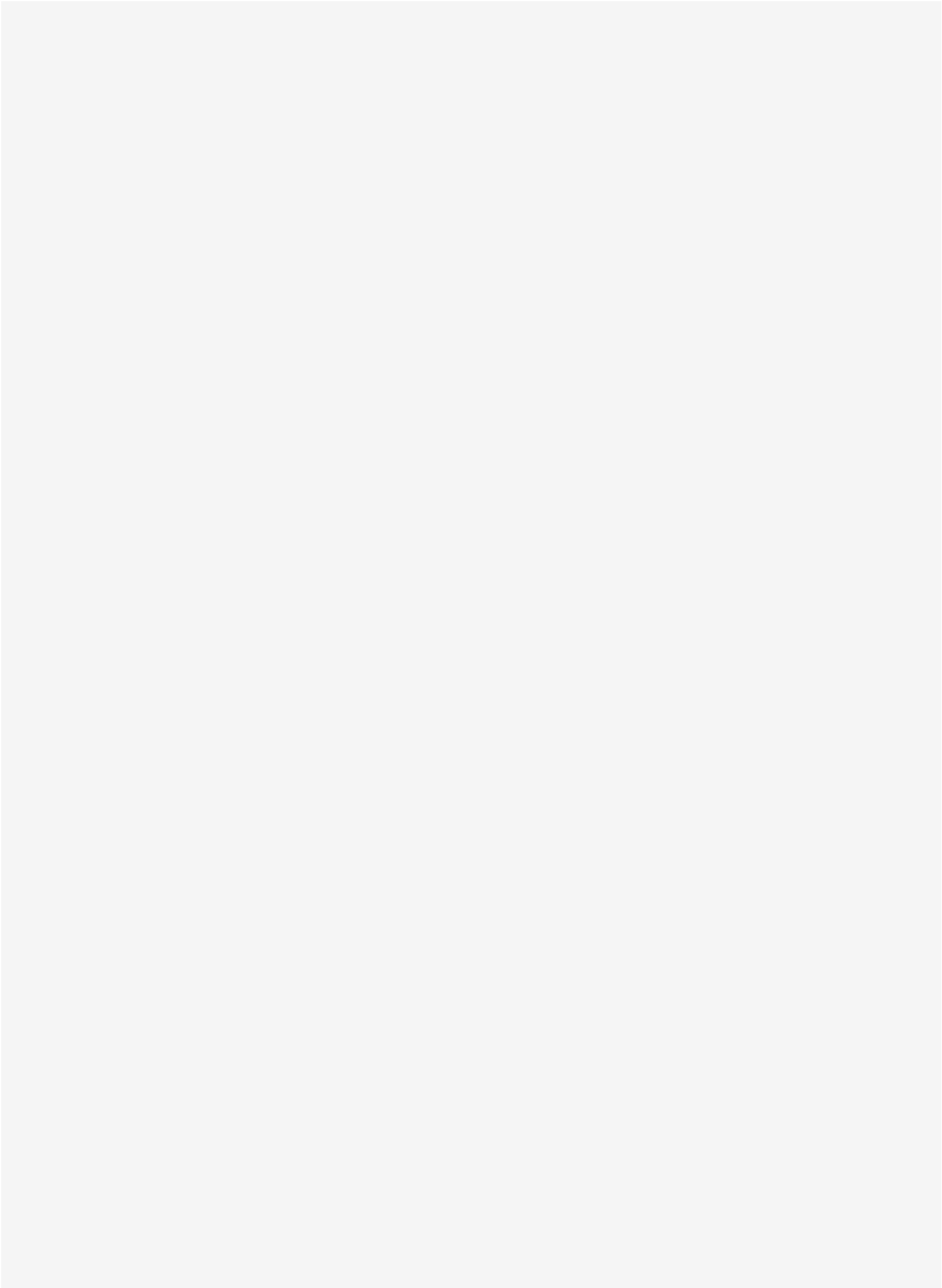



```
0097         if length(prop_index) == 1
0098             prop.(properties{prop_index}) = arg.(arg_fn{i});
0099         else
0100             error('Property ''%s'' does not exist or is ambiguous.', ...
0101                 arg_fn{i})
0102         end
0103     end
0104     arg_index = arg_index + 1;
0105 else
0106     error([
```

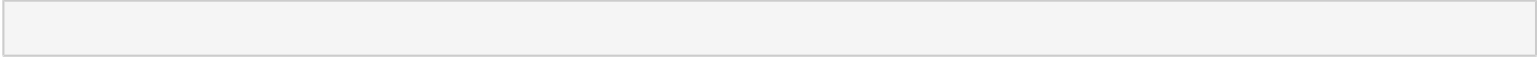
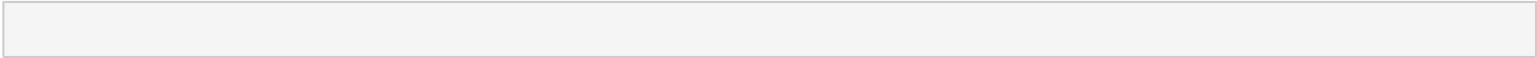



```
0206     'Callback',@clickpicks,...
0207     'Max',2);
0208
0209 openbut = uicontrol('Style','pushbutton',...
0210     'Position',[270 300 80 20],...
0211     'String','Open',...
0212     'Enable','off',...
0213     'Callback',@open);
0214 arrow = [2 2 2 2 2 2 2 2 1 2 2 2;,,@,@,@,@,@'Callback',@'pushbutton',...
0210     'Position'
```

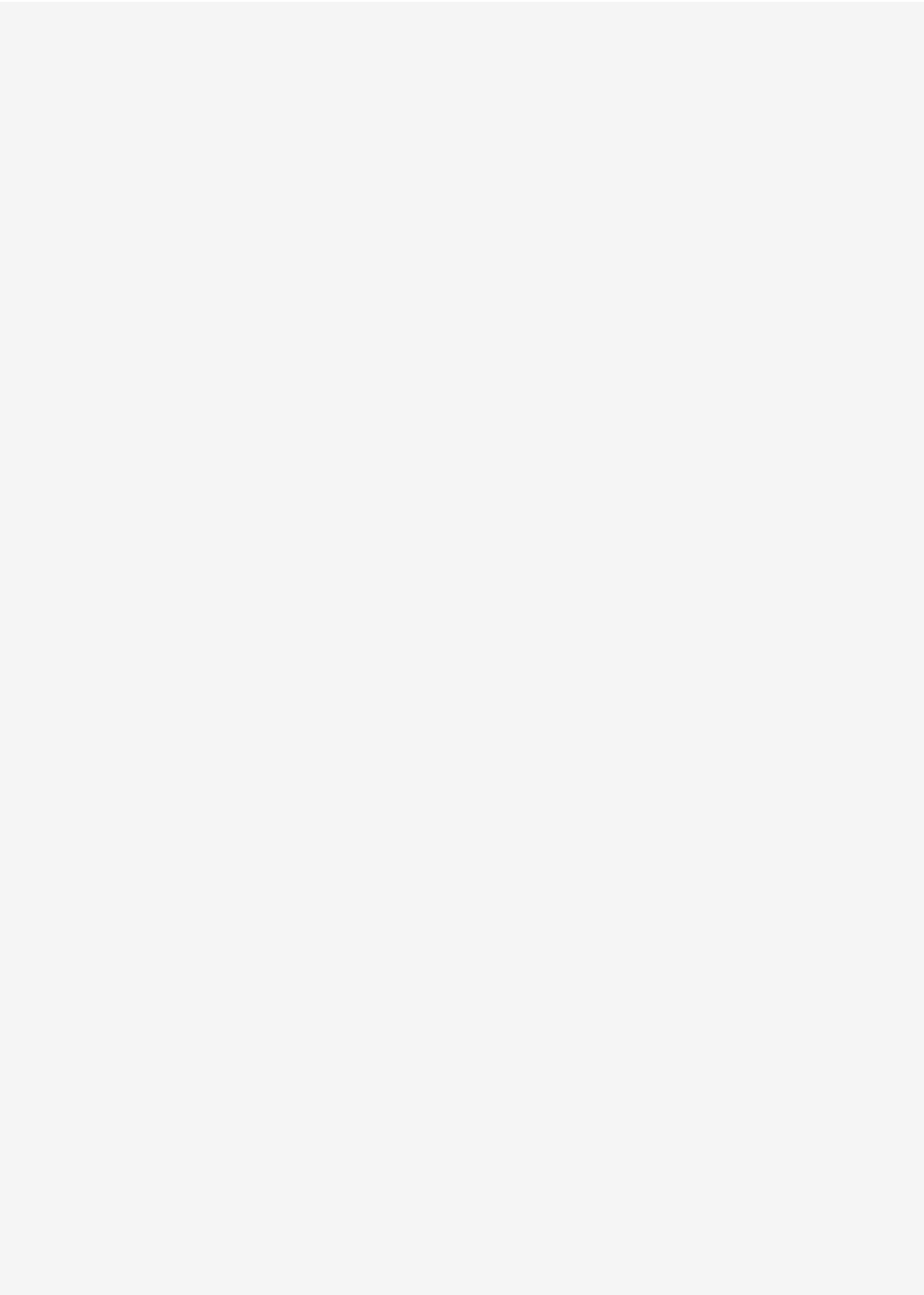









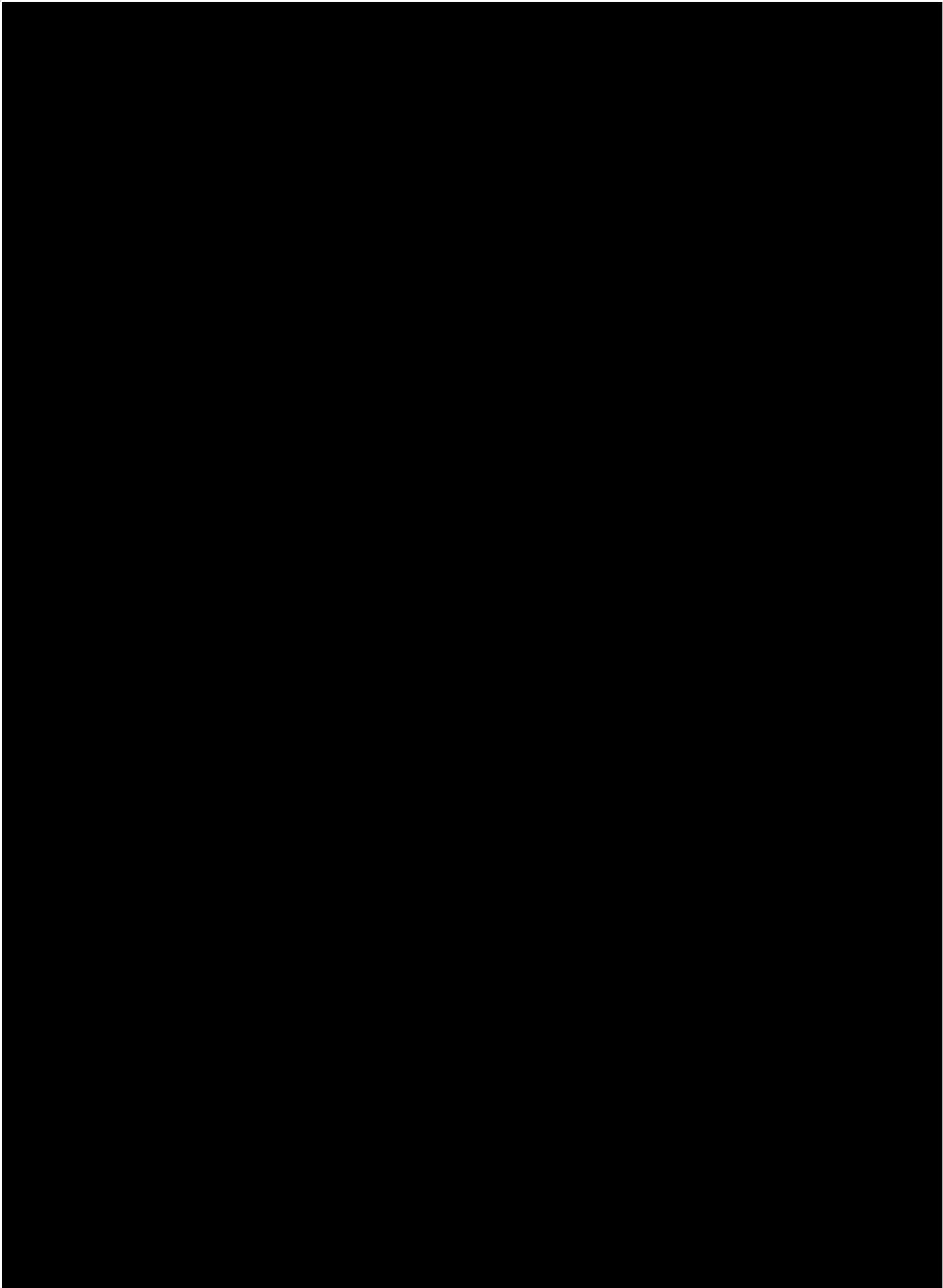

```
0203     panel(k) = fipanel( ...
0204         'Parent',htab, ...
0205         'Units','pixels', ...
0206         'Position',titlepos+[0,0,0,panelheight(k)*charsz(4)], ...
0207         'BackgroundColor',panelbackgroundcolor, ...
0208         'BorderType',panelbordertype, ...
0209         'Visible','off');
0210
0211     uimenu(cmenu, ...
0212         'Label',string{k}, ...
0213         'Callback',{@PopupTabCbk,k});
0214 end
0215
0216 uimenu(cmenu, 'D0Barator')
```



```

0318 if ~isnan(titlehighlightcolor)
0319     inset(2) = uicontrol( ...
0320         'Parent', htab, ...
0321         'Units', 'pixels', ...
0322         'BackgroundColor', titlehighlightcolor, ...
0323         'Style', 'text', ...
0324         'Visible', 'on');
0325     inset(3) = uicontrol( ...
0326         'Parent', htab, ...
0327         'Units', 'pixels', ...
0328         'BackgroundColor', 0.85*titlehighlightcolor, ...
0329         'Style', 'text', ...
0330         'Visible', 'on');
0331 end
0332
0333 TopBottomTabResizeCbk(status, []);
0334 set(status, 'ResizeFcn', @TopBottomTabResizeCbk);
0335 end
0336
0337 setappdata(htab, 'panels', panel);
0338 setappdata(htab, 'status', status);
0339 if ~isempty(createfcn)
0340     createfcn(htab, [], panel, status);
0341 end
0342
0343 if narginout, hh = htab; end
0344
0345
0346 %-----
0347 % Inner Functions for Popup 0
0348 %-----
0349 function Popup 0 Cbk(hobj, evdt, indx)
0350
0351     set(title, 'Value', 0);
0352     if indx>0
0353         indxvis = strcmp(get(panel(indx), 'Visible'), 'off');
0354         if indxvis
0355             set(panel(indx), 'Visible', 'on');
0356         else
0357             set(panel(indx), 'Visible', 'off');
0358         end
0359     end
0360     if isempty(strmatch('on', get(panel, 'Visible')))
0361         set(status, 'Visible', 'on');
0362     else
0363         set(status, 'Visible', 'off');
0364     end
0365
0366     currval = get(slider, 'Value');
0367     if currval<0
0368         set(slider, 'Value', 0);
0369         Popup 0 SliderCbk(slider, evdt);
0370     end
0371     titlepos = get(title(1), 'Position');
0372     addtolen = titlepos(2)+titlepos(4)+margins(4);
0373
0374     for k=1:nt0
0375         titlepos = get(title(k), 'Position');

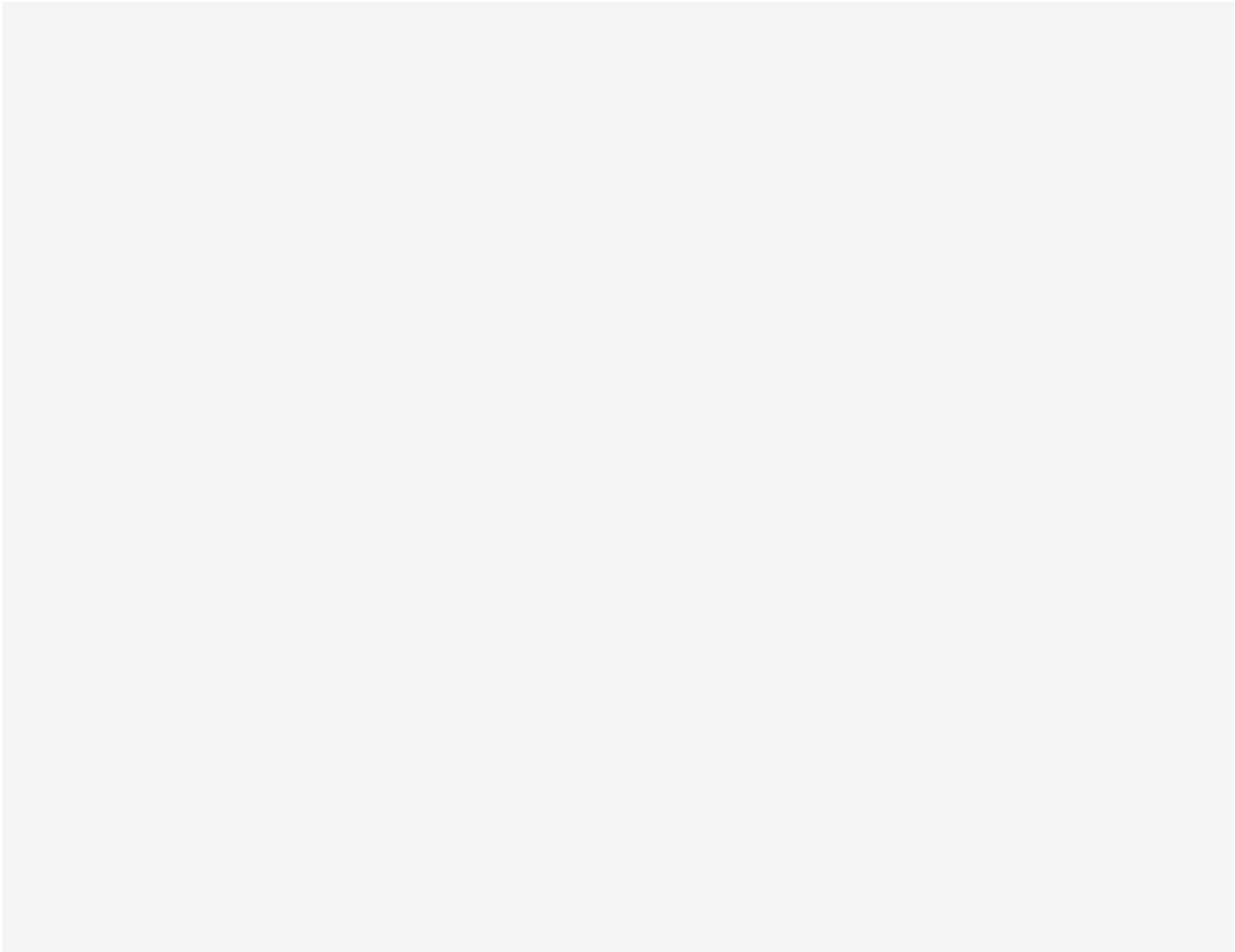
```


```

0009 % CALL:          classes = ltpda_classes()
0010 %
0011 % INPUTS:
0012 %
0013 % OUTPUTS:        classes - a cell array with a list of recognised LTPDA object
types
0014 %
0015 % VERSION:        $Id: ltpda_classes.m,v 1.1 2008/02/14 16:03:52 mauro Exp $
0016 %
0017 % HISTORY: 14-02-08 M Hueller
0018 %              Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022
0023 classes = {'ao',...
0024            'param',...
0025            'plist',...
0026            'cdata',...
0027            'tsdata',...
0028            'fsdata',...
0029            'xydata',...
0030            'xyzdata',...
0031            'specwin',...
0032            'mfir',...
0033            'miir',...
0034            'pzmodel',...
0035            'pole',...
0036            'zero',...
0037            'provenance',...
0038            'time',...
0039            'timeformat',...
0040            'timespan',...
0041

```


```
0287     mtable = get(hObject,'userdata');
0288     jtable = mtable.getTable;
0289
0290     % Stop any current editing
0291     stopEditing(jtable);
0292
0293     % Insert the new row immediately before the current row
0294     newRowData = cell(1,mtable.getNumColumns); % empty data
0295     mtable.getTableModel.insertRow(max(0,jtable.getSelectedRow), newRowData);
0296 %catch
0297
```



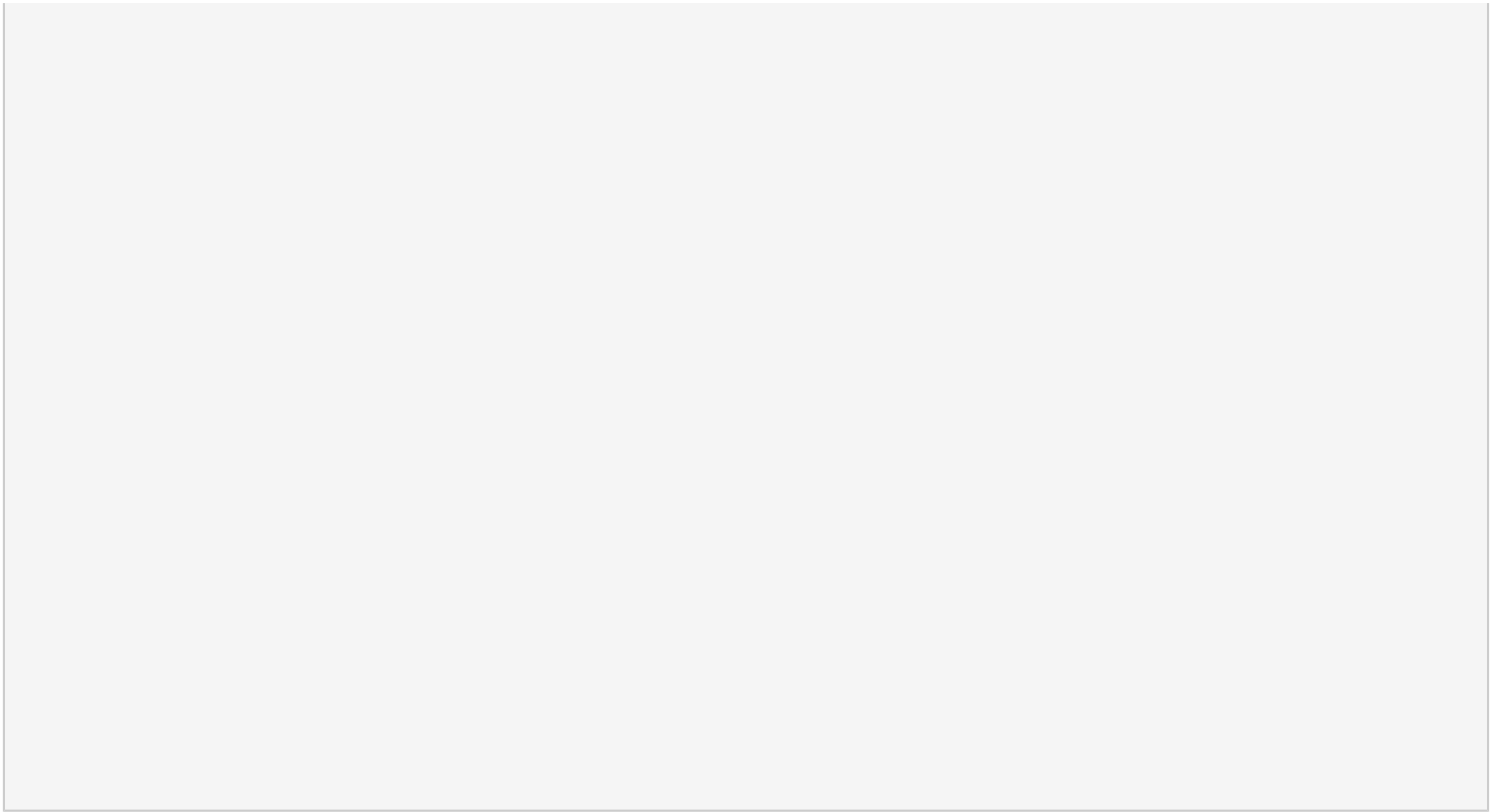
```
0450
0451      % Close the workbook
0452      invoke(hExcel.ActiveWindow, 'close', false);
0453(% C 1 0 0 0 rgcatchok)Tj 0 0 0 rg T* (0451      )Tj 0.133 0.545 0.133 rg (Insert
0751
```



```

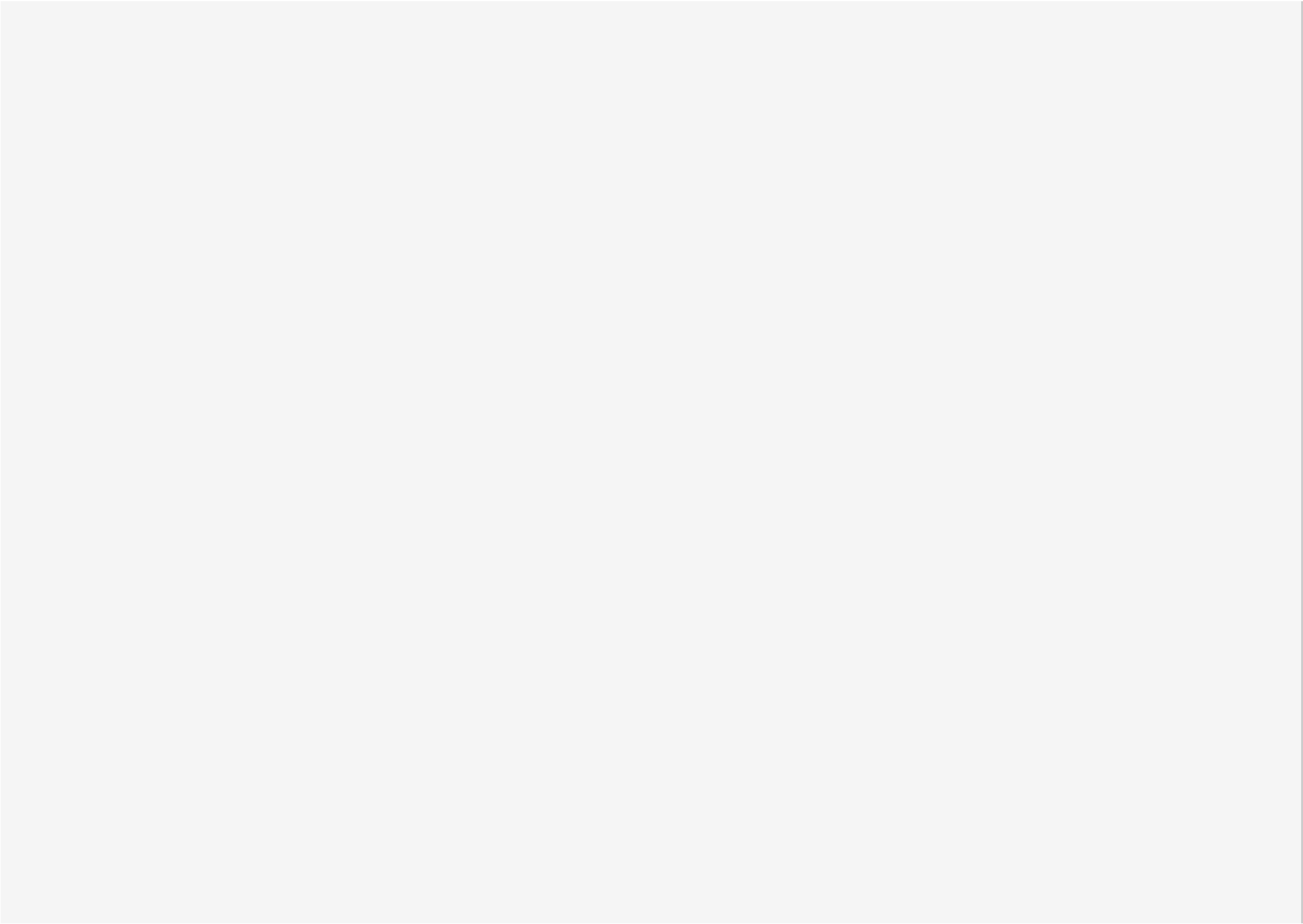
0008 %
0009 % CALL:          ltpda_constructor_helper
0010 %
0011 %
0012 % VERSION:       $Id: ltpda_constructor_helper.m,v 1.3 2008/03/17 20:48:47
hewitson Exp $
0013 %
0014 % HISTORY: 07-03-08 M Hewitson
0015 %           Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019
0020 %% Check if I exist already
0021 id = findobj('Tag', 'LTPDAconstructor_helper');
0022 if ~isempty(id)
0023     figure(id)
0024     return
0025 end
0026
0027 %% Some initial setup
0028
0029 Gproperties.Gcol      = [240 240 240]/255;
0030 Gproperties.Gwidth    = 500;
0031 Gproperties.Gheight   = 300;
0032 Gproperties.Gborder   = 10;
0033 fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
0034
0035 Gproperties.Screen    = get(0, 'screensize');
0036 Gproperties.Gposition = [150 ...
0037     150 ...
0038     Gproperties.Gwidth...
0039     Gproperties.Gheight];
0040
0041 % Initialize and hide the GUI as it is being constructed.
0042 mainfig = figure('Name', 'LTPDA Constructor Helper', ...
0043     'NumberTitle', 'off', ...
0044     'Visible', 'off', ...
0045     'Position', Gproperties.Gposition, ...
0046     'Color', Gproperties.Gcol, ...
0047     'Resize', 'off', ...
0048     'ToolBar', 'none', ...
0049     'NextPlot', 'new', ...
0050     'MenuBar', 'none', ...
0051     'Tag', 'LTPDAconstructor_helper');
0052
0053 % Set mainfig callbacks
0054 set(mainfig, 'CloseRequestFcn', {@ltpda_constructor_helper_close, mainfig});

```

Description of progressBar

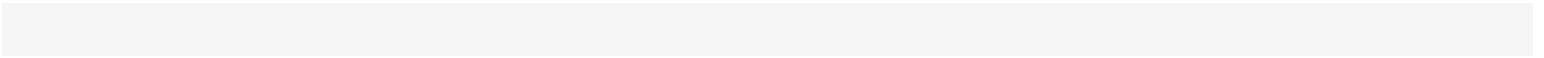


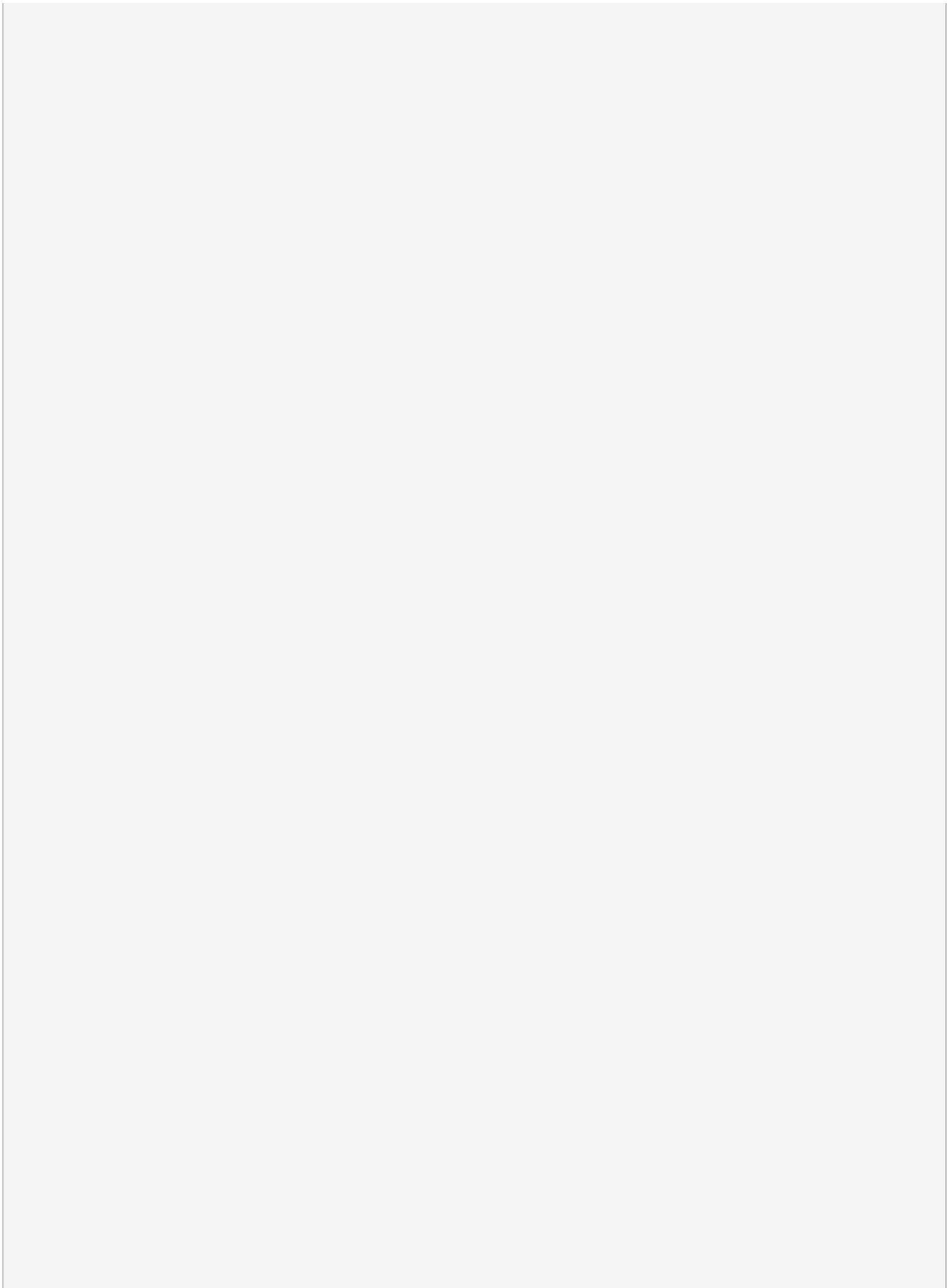



```
0020     set(lh, 'Value', 1);
0021     set(lh, 'String', {'Default'});
0022 else
0023     set(lh, 'Value', 1);
0024     set(lh, 'String', sets);
0025 end
0026
```

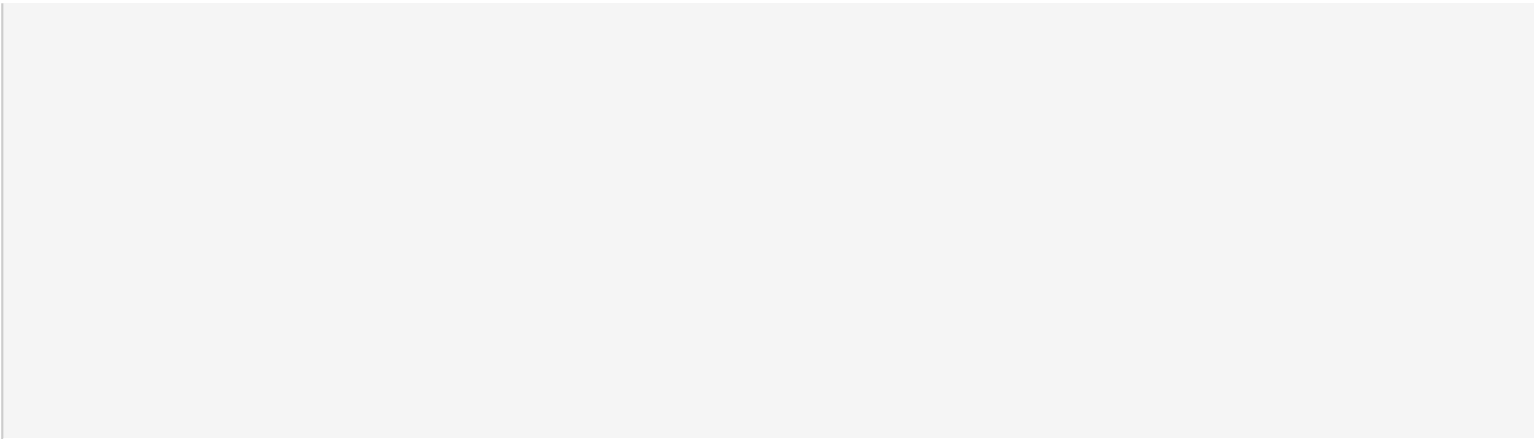
Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003


```
0018     pl = getappdata(mainfig, 'plist');
0019 else
0020     eval(sprintf('pl = %s(%s, ''Params'', ''%s'');', cl, cl, set)79 r
```





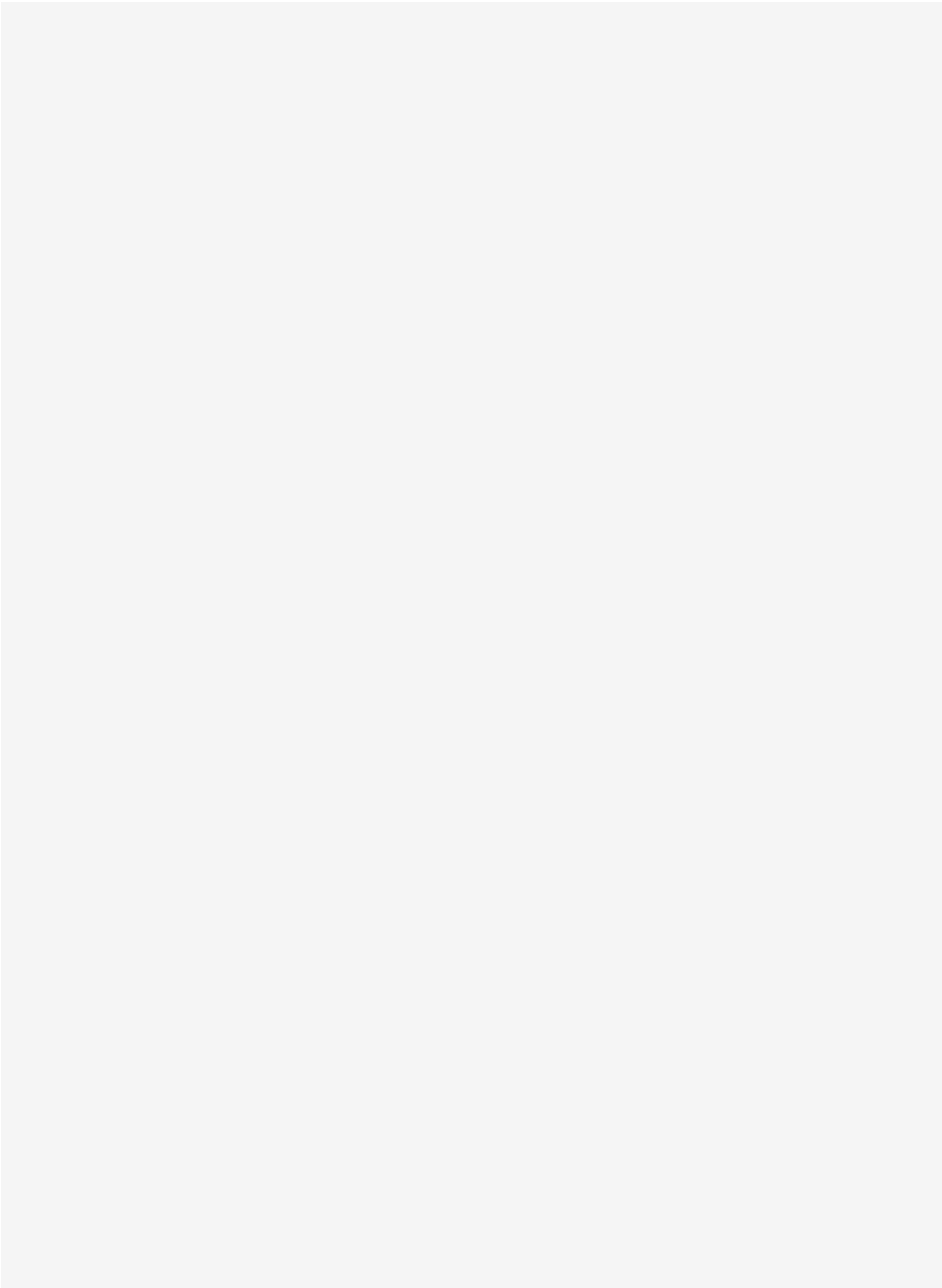
```
0130 %   if get(zeros, 'q') > 0.5
0131 %       [ai,bi] = cz2iir(zero, fs);
0132 %   else
0133 %       [ai,bi] = rz2iir(zero, fs);
0134
```

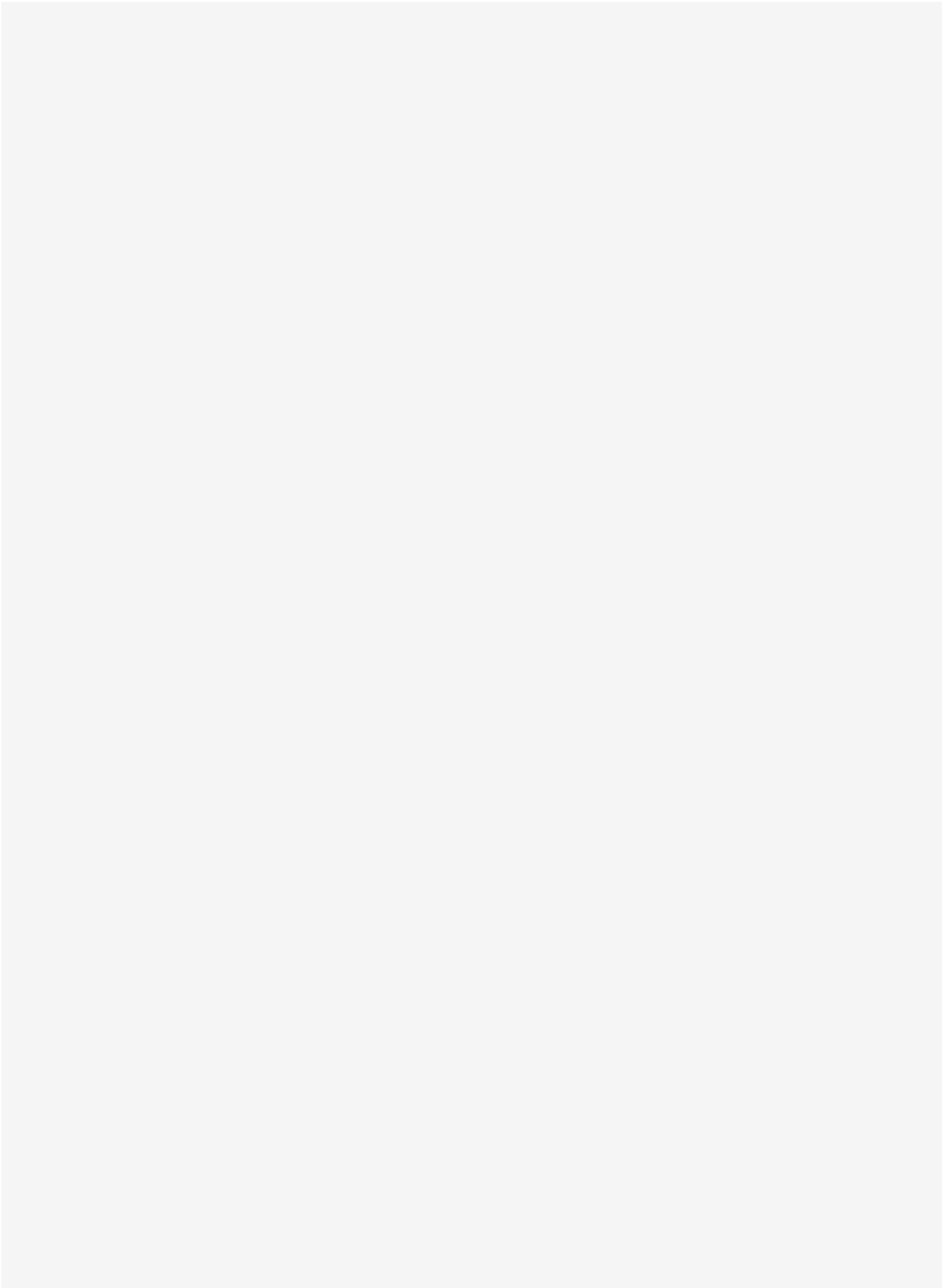



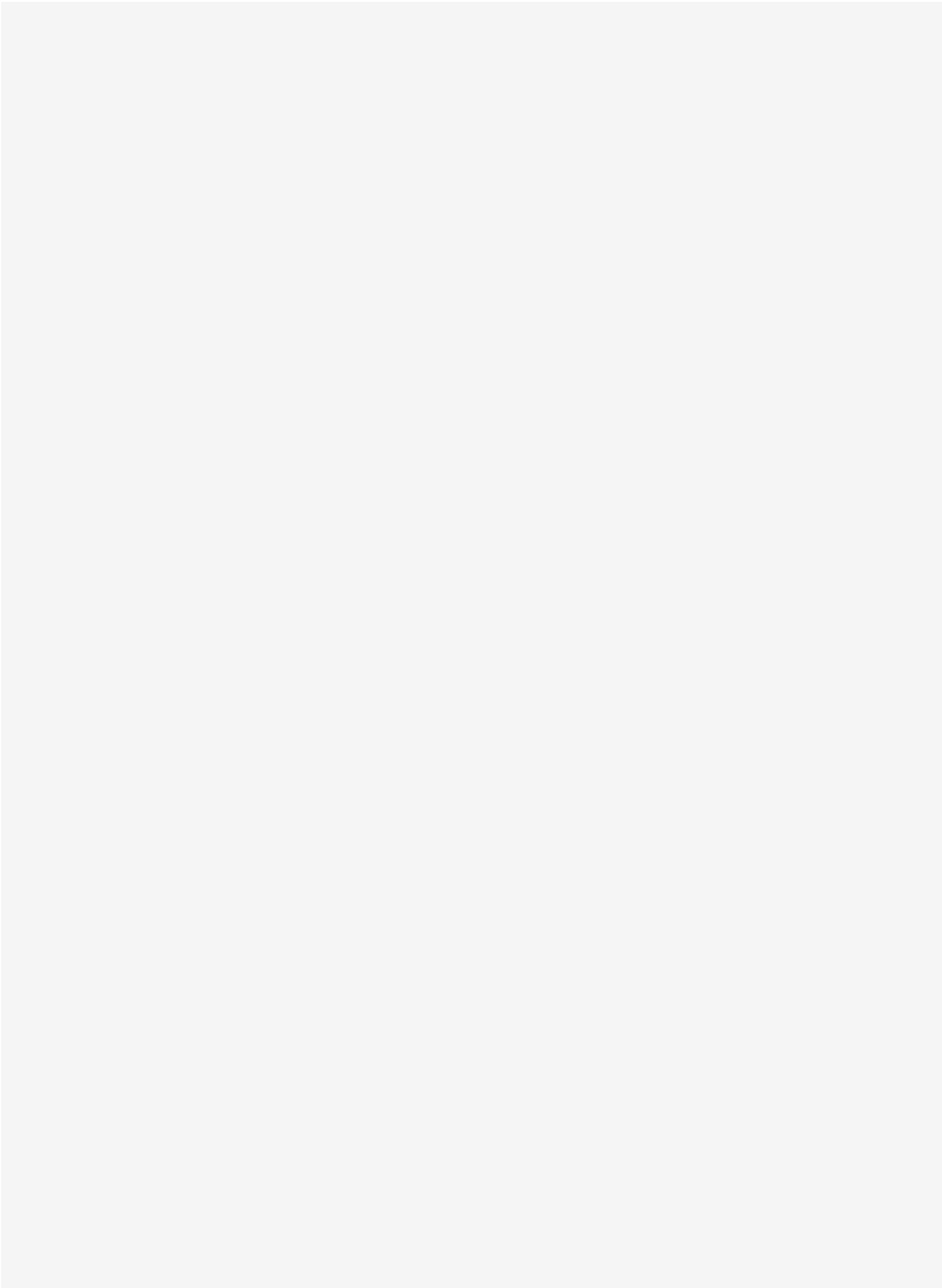
[Home](#) > [m](#) > [gui](#) > [pzmodel_designer](#) > pzmodel_helper.m

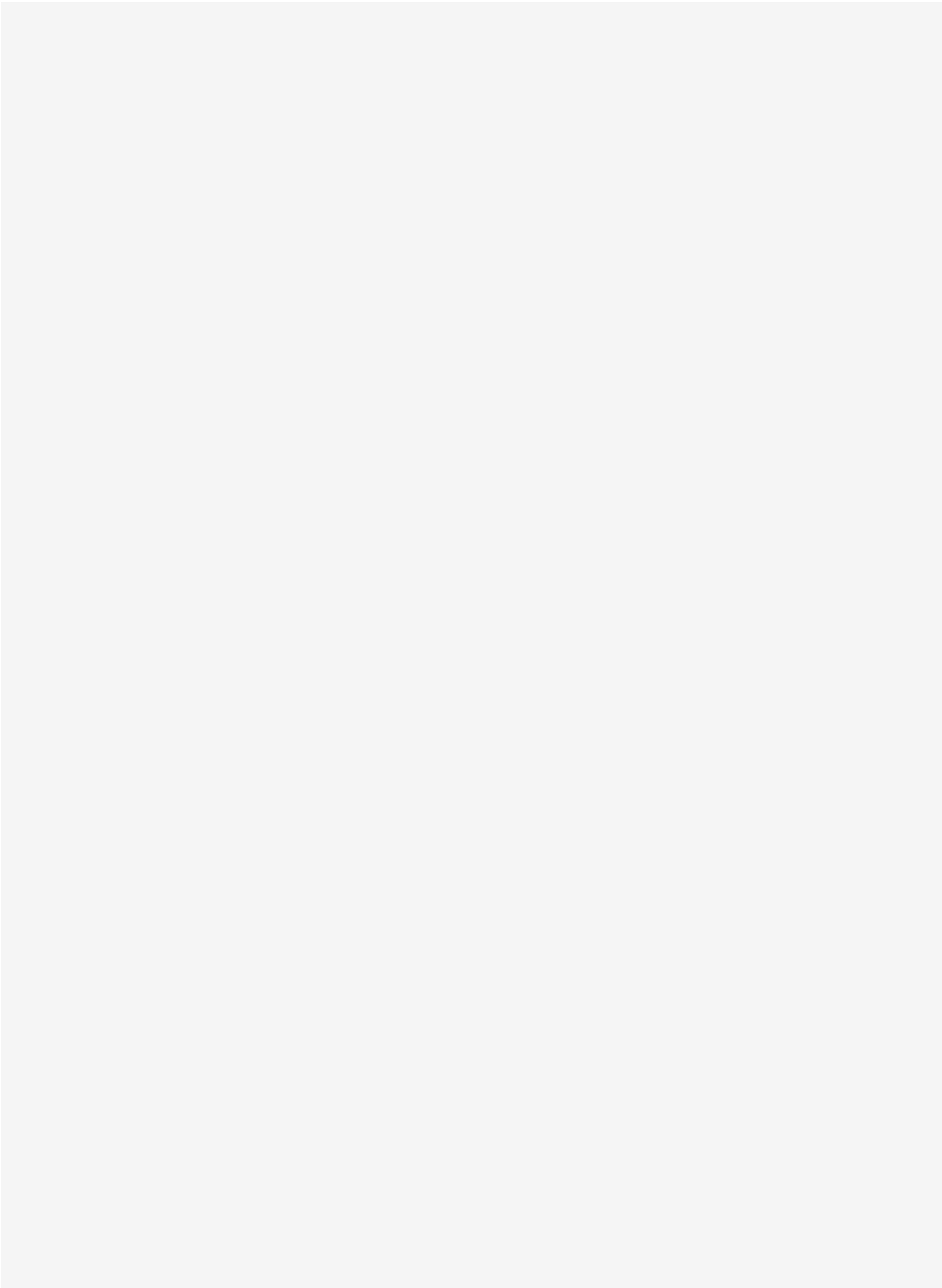
pzmodel_helper

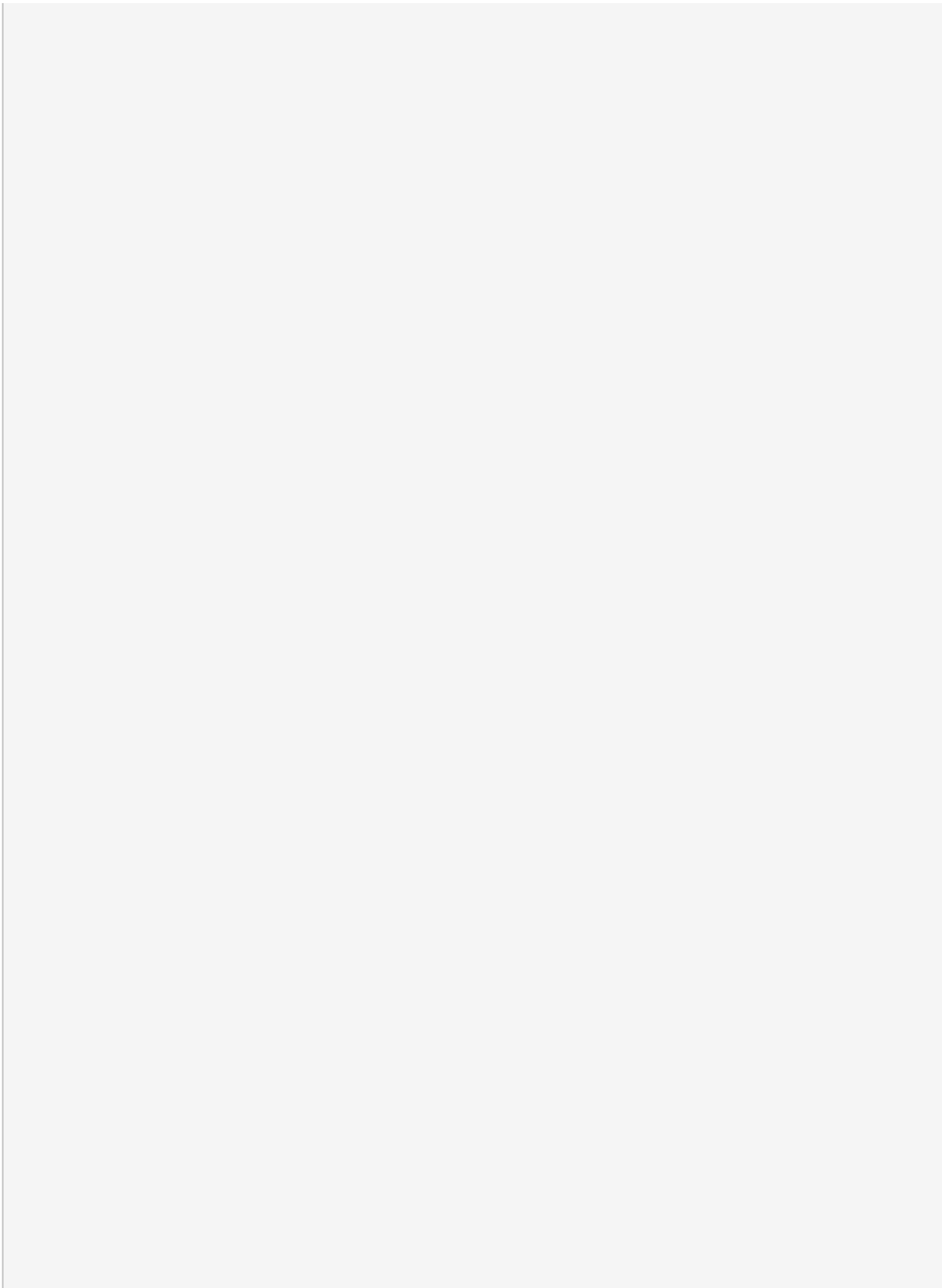
PURPOSE







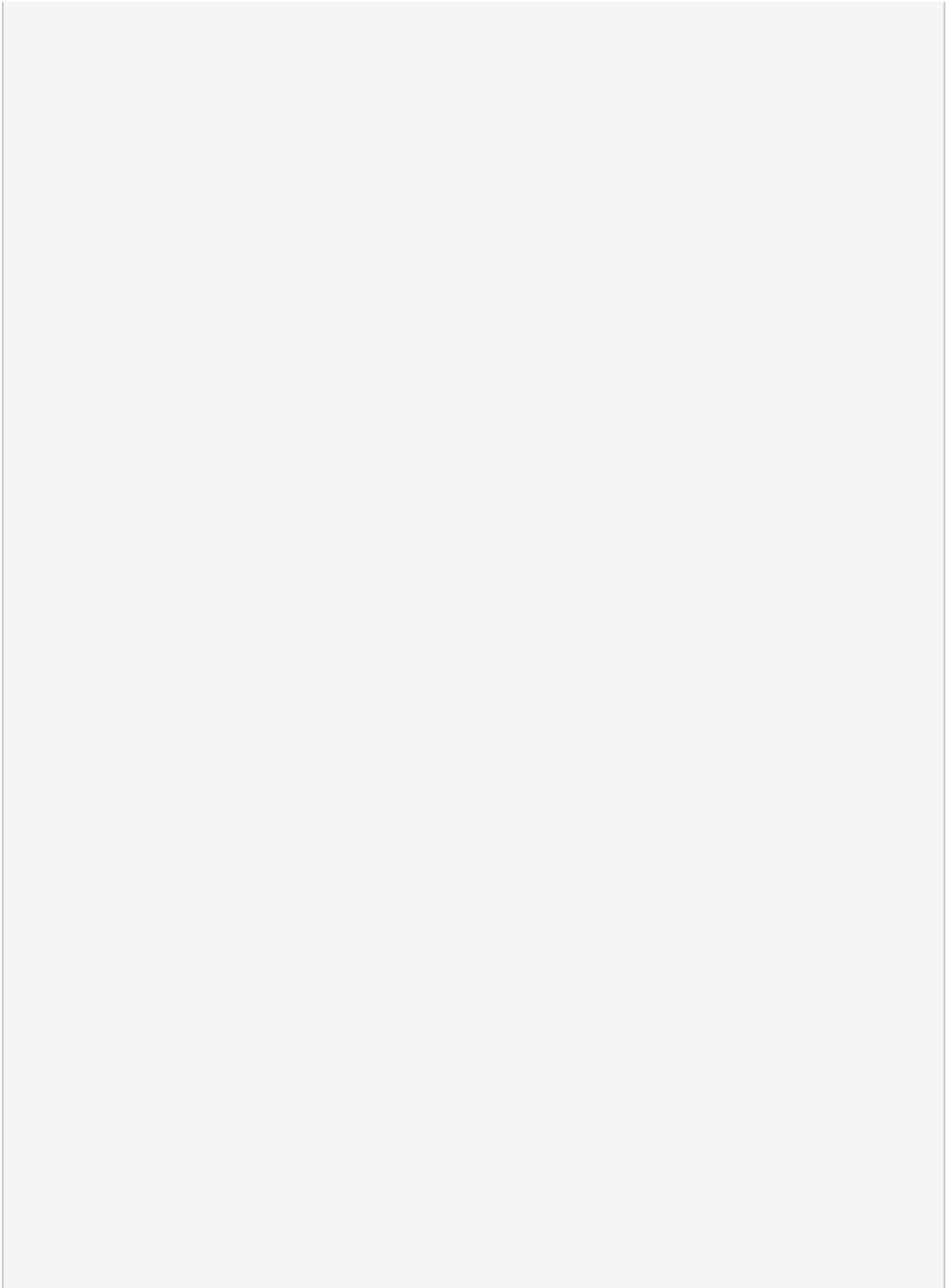




```
0413 if ispc && isequal(get(hObject,
```



```
0328         end
0329
0330     end
0331     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0332     function f_disp_menu1(h, ev)
0333         set(txt5, 'Visible', 'off'),
```


```

0674
0675     for i=1:length(fnames)
0676         count = count + 1;
0677         x = getfield(val,fnames{i});
0678
0679         if isa(x, 'ao')
0680             iconpath =[pth,'analysis_object.gif'];
0681         elsej T* (0680             iconpath =[pth,)Tj 0.tsdata')
0682             iconpath =[pth,'ts_data.gif'];
0683         elsej T* (0680             iconpath =[pth,)Tj 0.fsdata')
0684             iconpath =[pth,'fs_data.gif'];
0685         elsej T* (0680             iconpath =[pth,)Tj 0.xydata')
0686             iconpath =[pth,'xy_data.gif'];
0687         elsej T* (0680             iconpath =[pth,)Tj 0.cdata')
0688             iconpath =[pth,'c_data.gif'];
0689         elsej T* (0680             iconpath =[pth,)Tj 0.history')
0690             iconpath =[pth,'history.gif'];
0691         elsej T* (0680             iconpath =[pth,)Tj 0.plist')
0692             iconpath =[pth,'plist.gif'];
0693 (];)Tj T* 94             elsej T* (0680             struct    ))Tj T* (0695

```


--

--

--


```
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);
```



```
0068      'Title',{ 'Connection','Submit','Query','Retrieve'},...%, 'Quick Look'},...
0069      'HorizontalAlignment','left',...
0070      'FontWeight','bold',...
0071      'TitleBackgroundColor',Gproperties.Gcol,...
0072      'TitleForegroundColor','k',...
007398 0 0 rg ('k')Tj 0 0 0 rg 900hhightColor', [1 0 0],...
007498 0 0 rg ('k')Tj 0 0 PanelBackgroundColor',Gproperties.Gcol,...
007598 0 0 rg ('k')Tj 0 0 PanelBopt 0 0 rg (,)Tj 598 0 0 rg ('k')Tj 0 0 PanelBopt
```

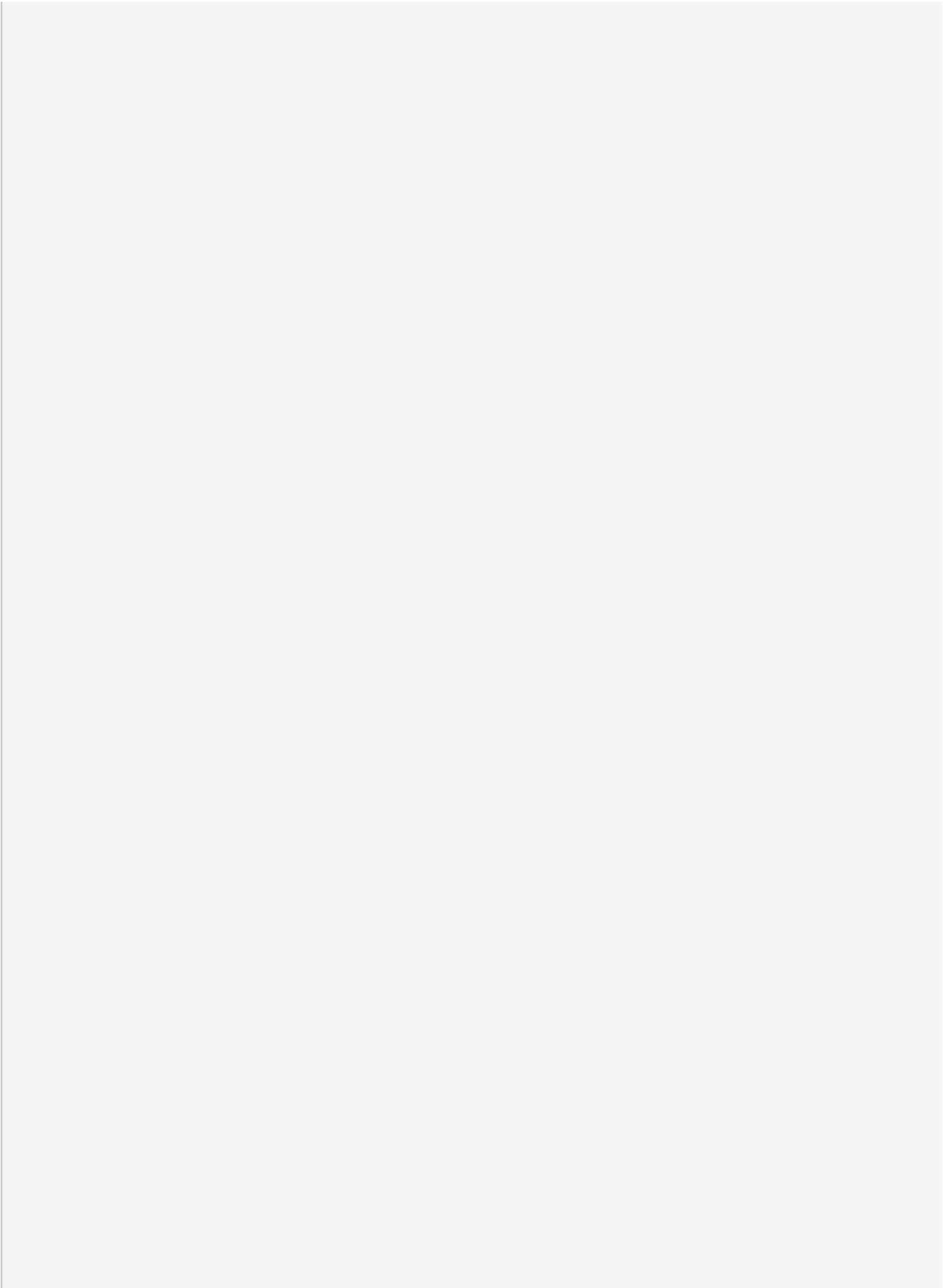
```

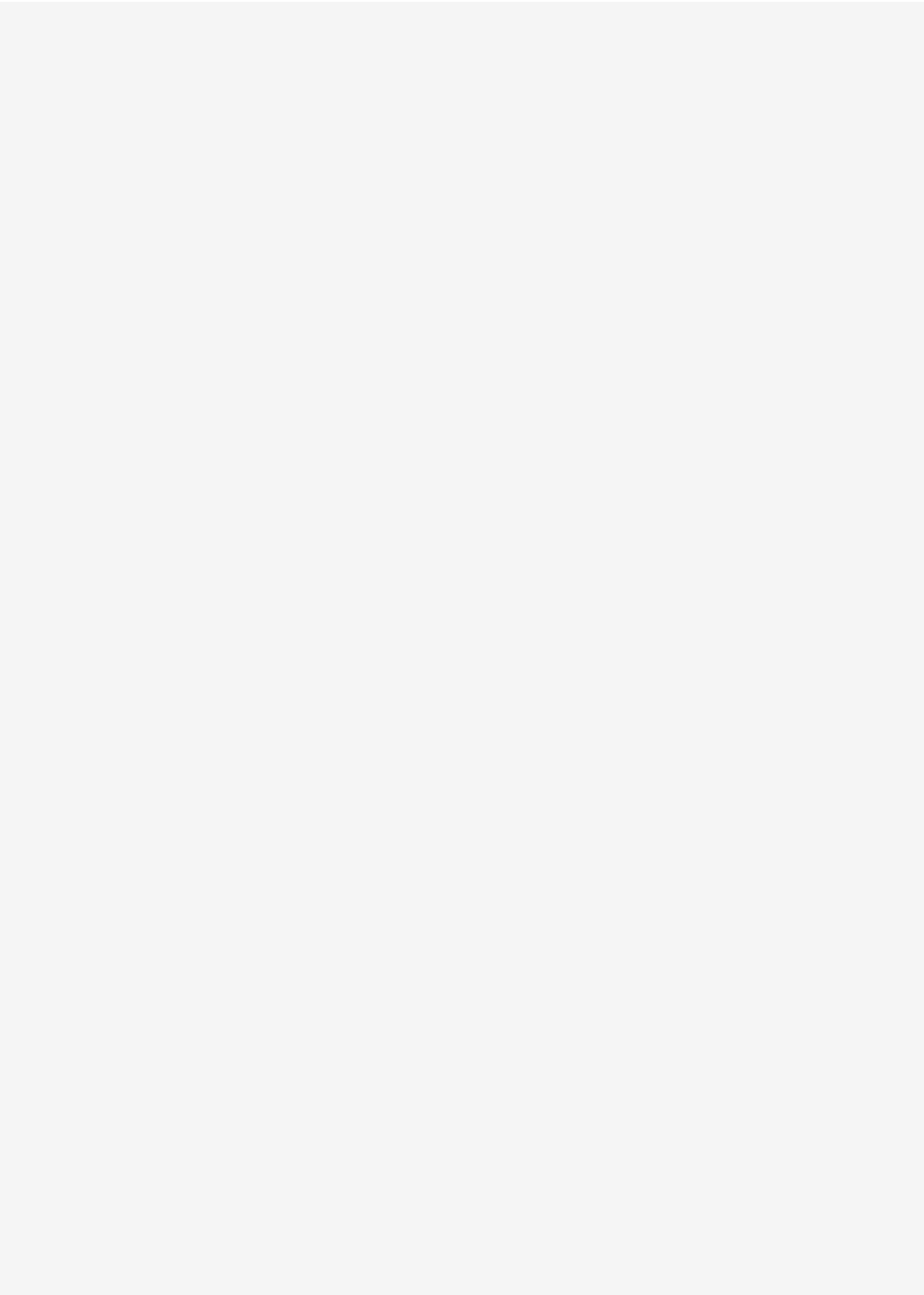
0124
0125 h = findobj(mainfig, 'Tag', 'statusTxt');
0126 function panel = create_quicklook_panel(panel, Gproperties)
0127 get(h, 'String')
0128 Ppos = get(panel, 'Position');
0129 mainfig = findobj('Tag', 'LTPDAREpomainfig');
0130 %-----
0131 %-create the quick look panel
0132 $left = 10;
0133 Sheight = 20;
0141 Sbottom = Ppos(2) + Ppos(4) - 20 - Gproperties.Gborder - Sheight;
0142 Swidth = 200;
0143 fontsize = getappdata(0,

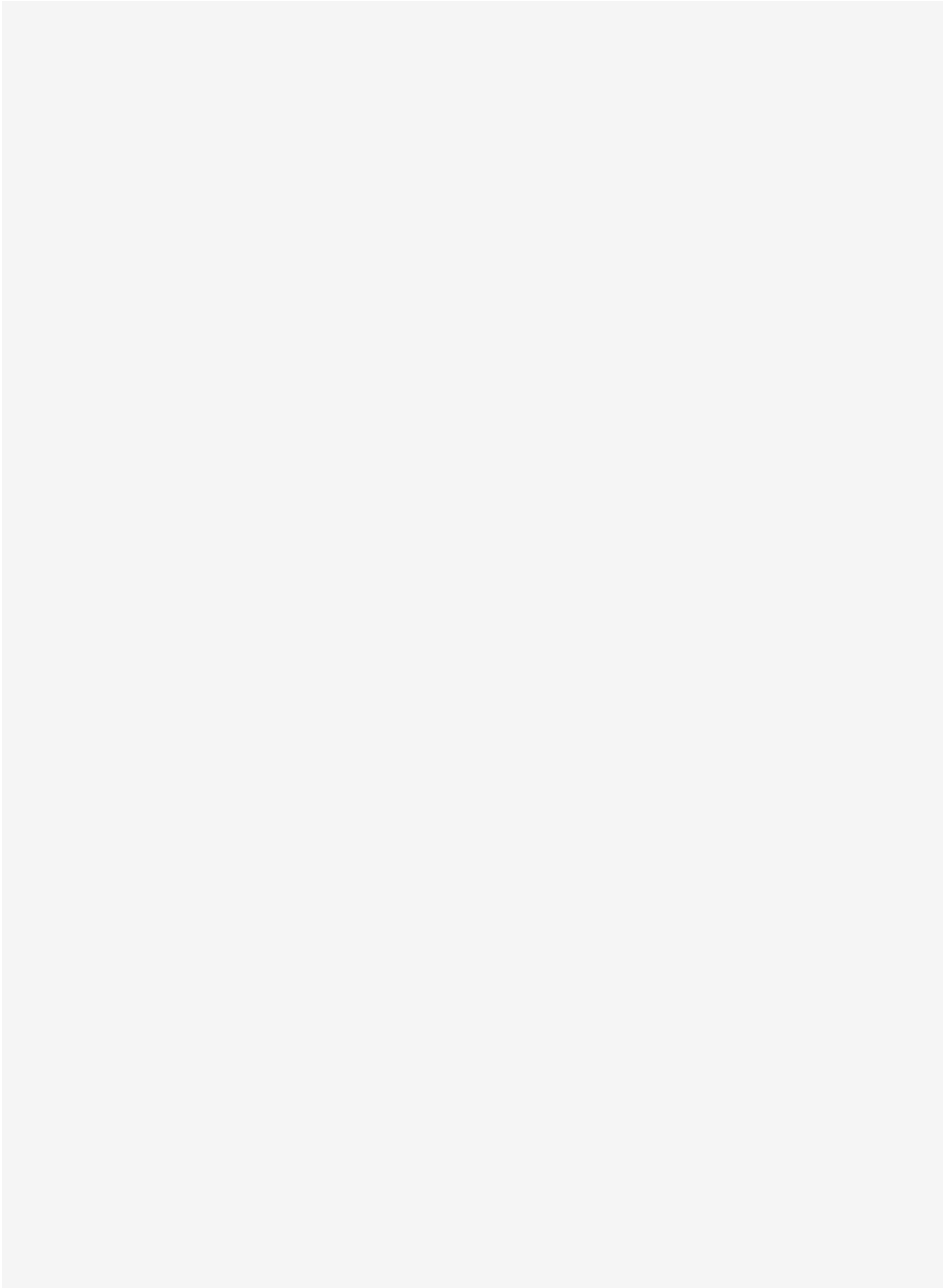
```



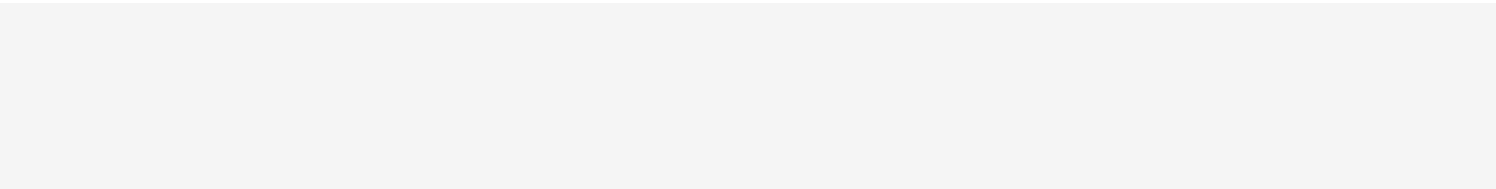
```
0236 %------ Help panel
0237 pos [ .ut(lbh, 'Position');
0238 Sleft  [ pos(1) + pos(3) + 10;
0239 Sheight [ 150;
0240 Sbottom [ pos(2);
0241 Swidth  [ 550;
0242 sth [ 3/j T* 2Tj ol(l
```

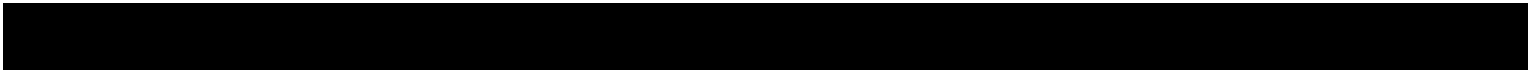








```
0740 %- - Text string
0741 sth = uicontrol(panel, 'Style', 'text', ...
0742     'String', 'Server hostname', ...
0743     'FontSize', fontsize, ...
0744     'BackgroundColor', Gproperties.Gcol, ...
0745     'HorizontalAlignment', 'left', ...
0746     'Position', [Tleft+Twidht+10 Tbottom 120 Theight]);
0741
0748
```





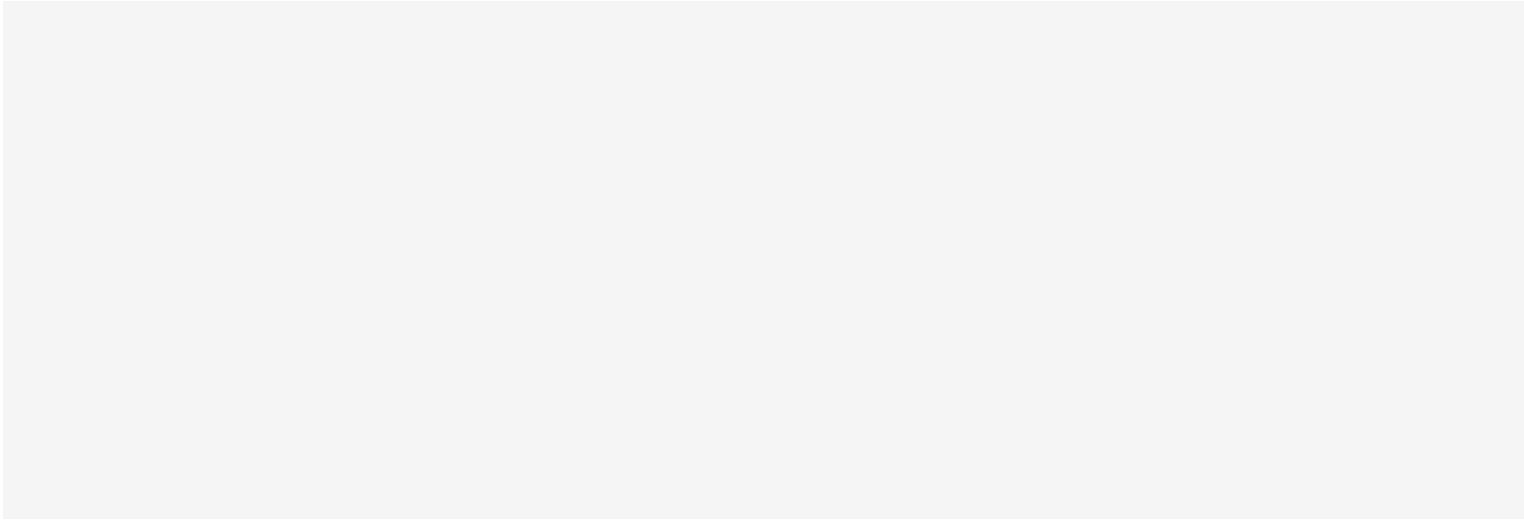
```
0013 v(idxy8] 2-v(idxy;  
0014  
0015  
0016 % Make window struct  
0017 w.name      = 'Bartlett';  
0018 w.alpha     = 0;  
0019 w.psll      = 26.5;  
0020 w.rov       = 50;  
0021 w.nenbw     = 1.3333;  
0022 w.w3db      = 1.2736;  
0023 w.flatness  = -1.8242;  
0024 w.ws        = sum(vy;  
0025 w.ws2       = sum(v.*vy;  
0026 w.win       = v;  
0027  
0028  
0029  
0030 % END
```



```
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
```

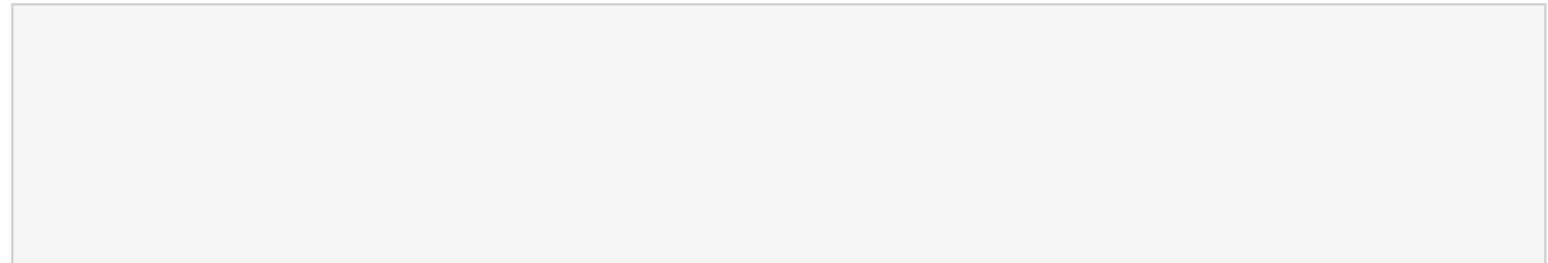
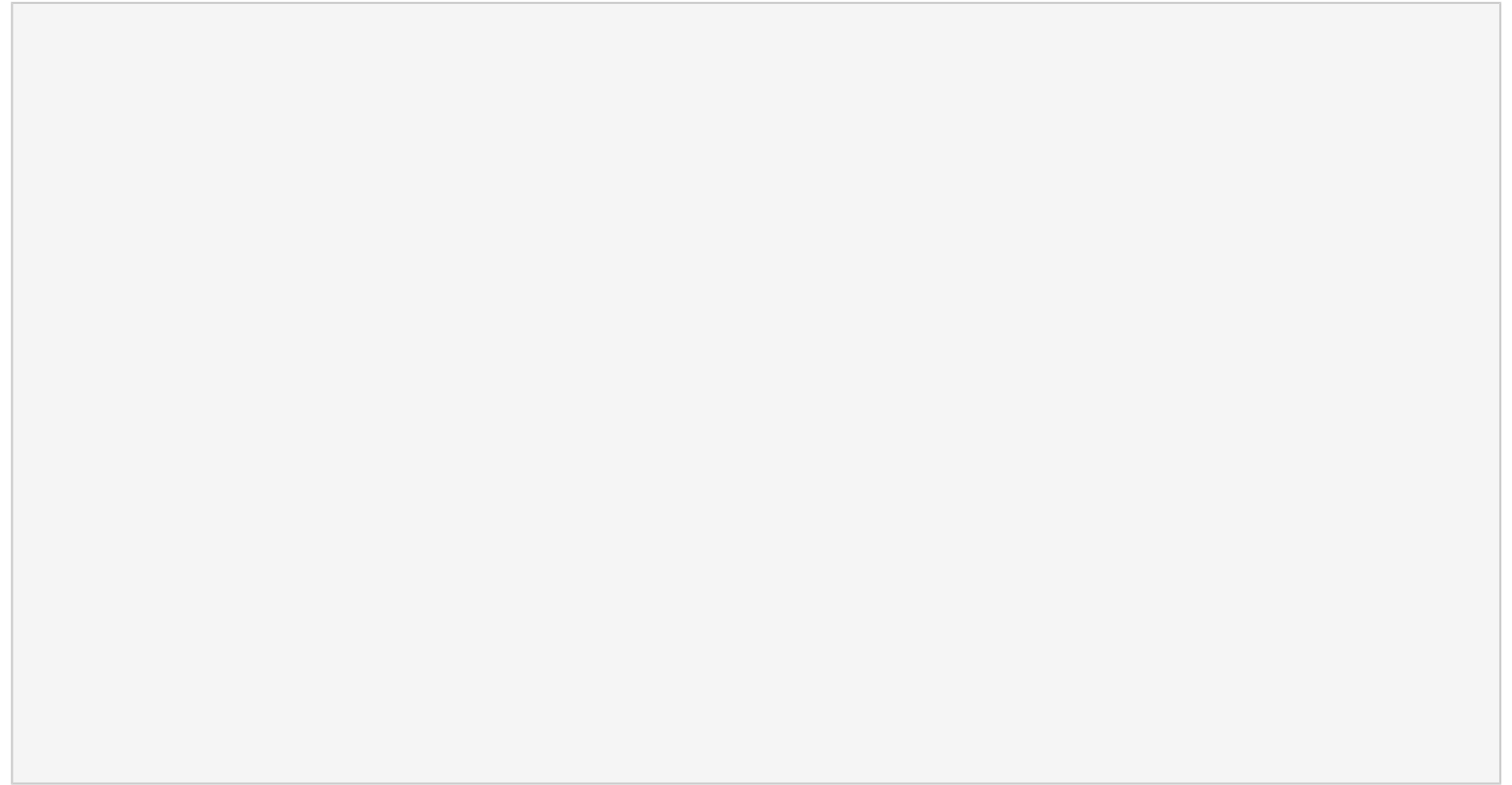
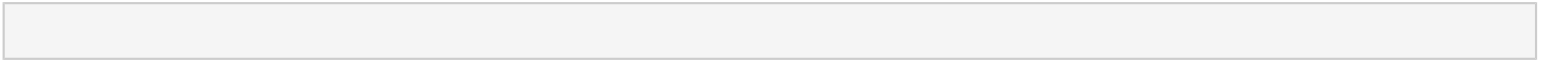


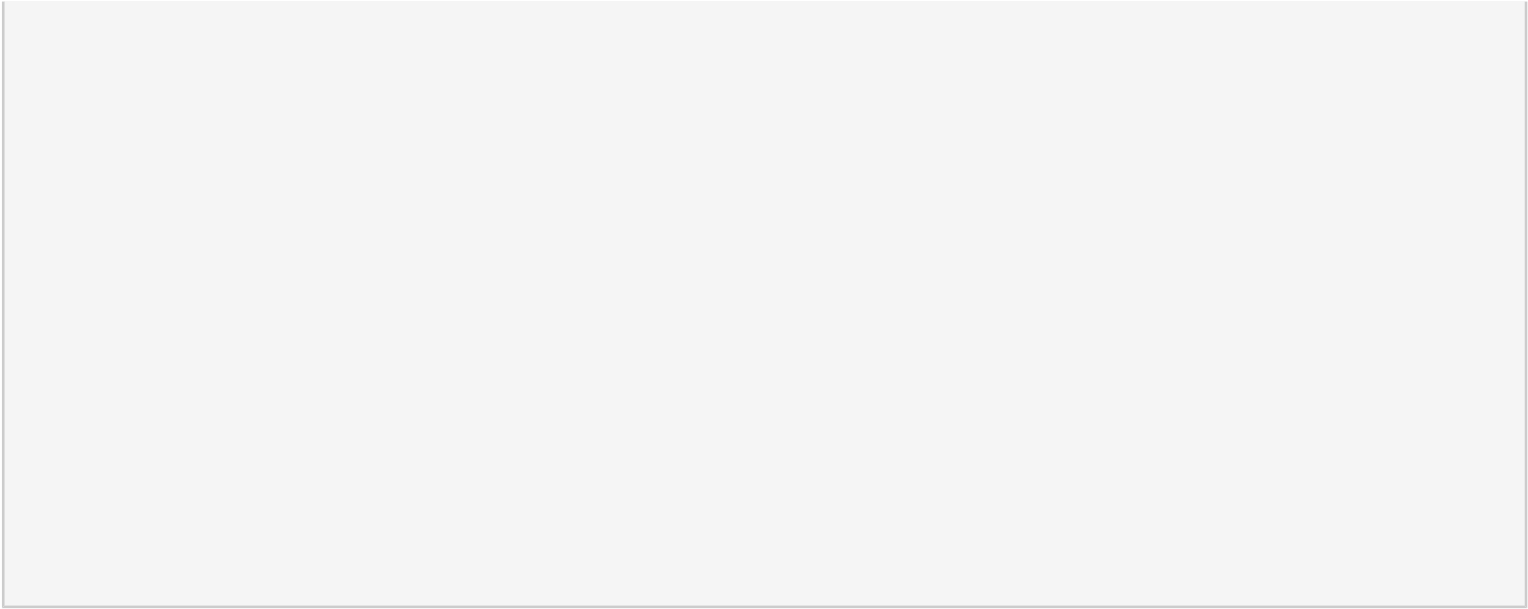
```
0013 % Make window struct
0014 w.name      = 'FTNI';
0015 w.alpha     = 0;
0016 w.psll      = 44.4;
```

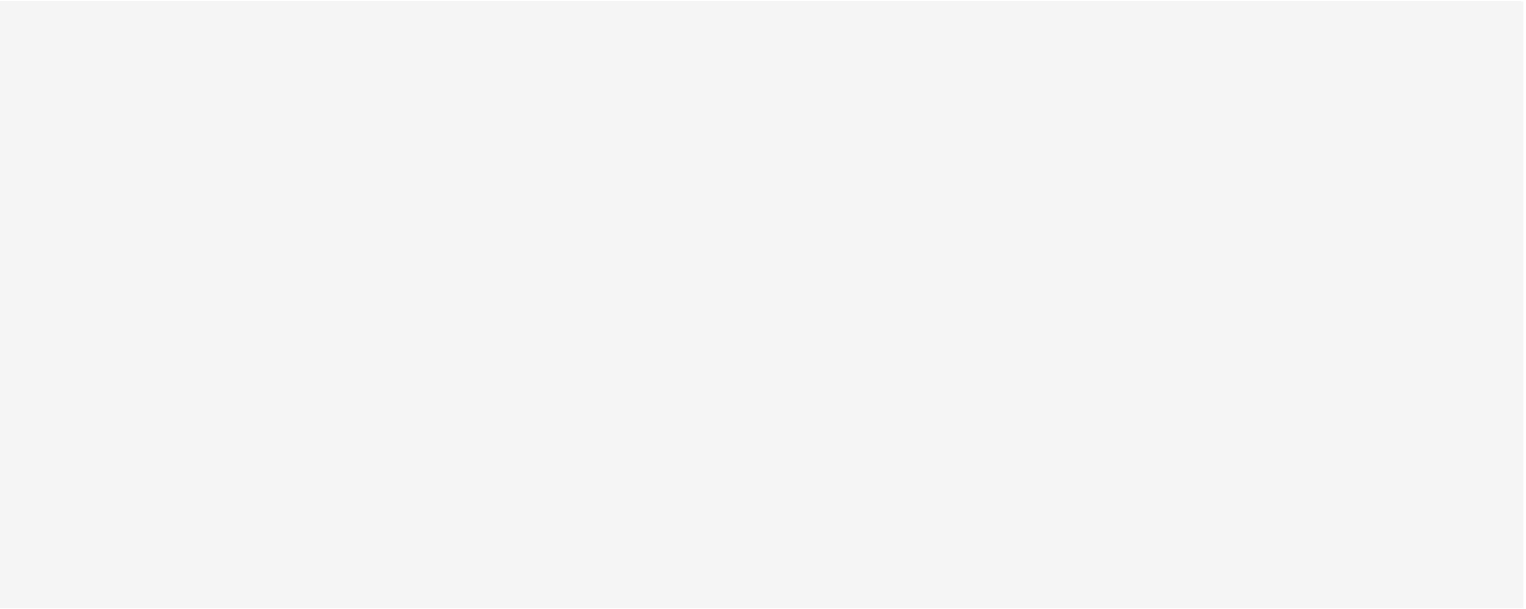





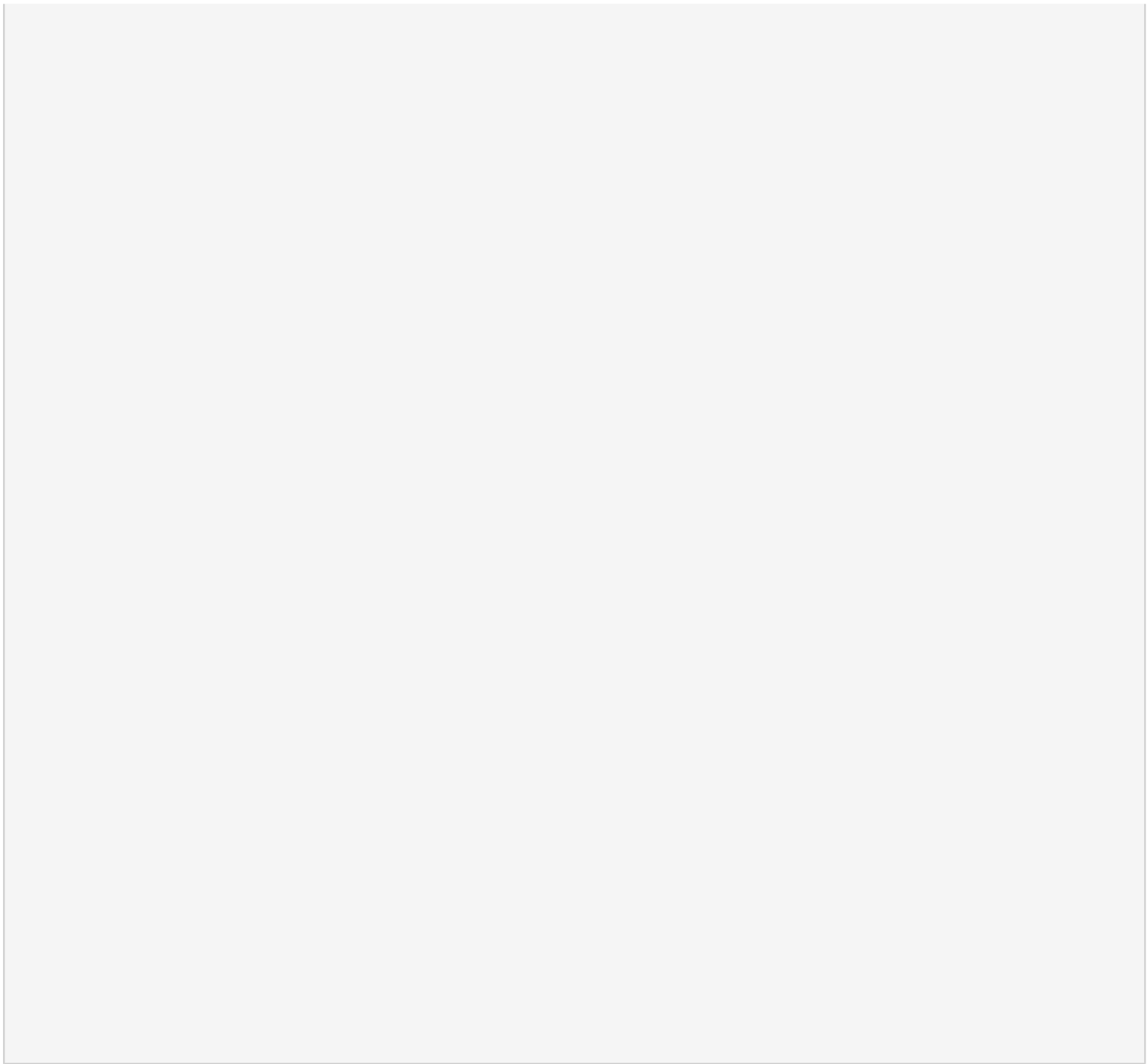
```
0010 % INPUTS:      a      - vector of double
0011 %
0012 % OUTPUTS:      index - index of the maximum
0013 %               max   - value of the maximum
0014 %
0015 % VERSION: $Id: ltpda_getmax.m,v 1.5 2007/07/13 12:17:39 ingo Exp $
0016 %
0017 % HISTORY: 26-01-2007 M Hewitson
0018 %           Creation
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
```

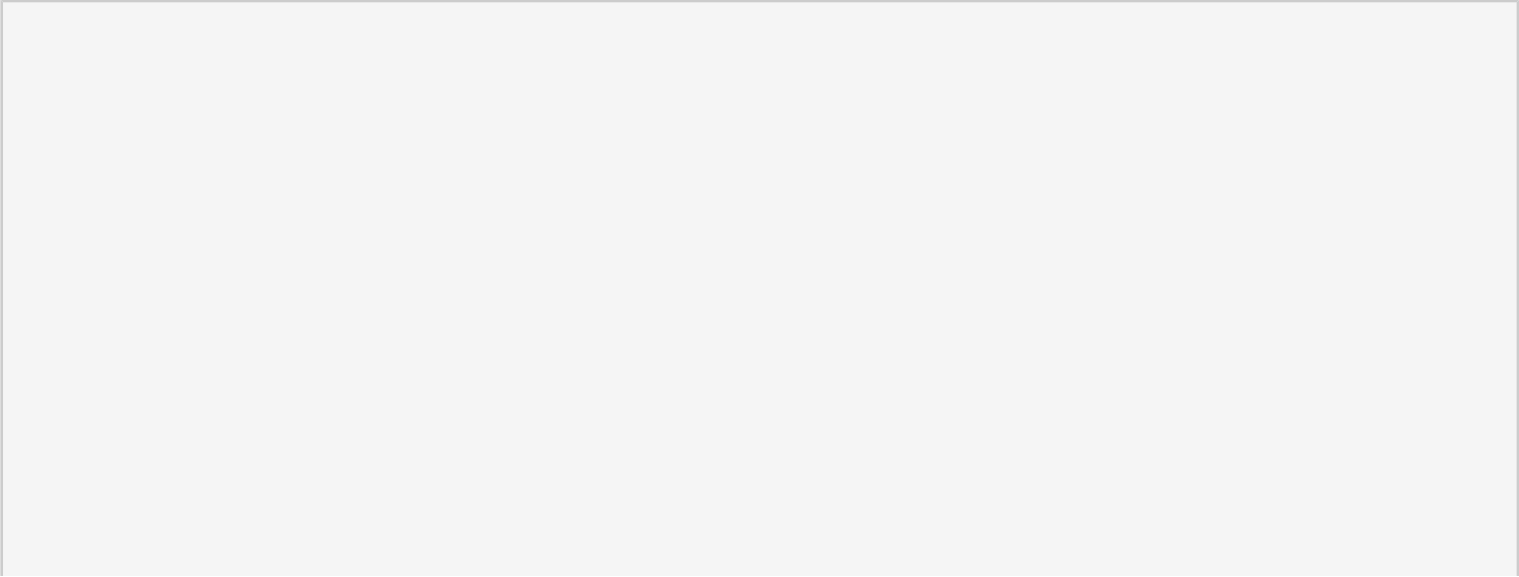
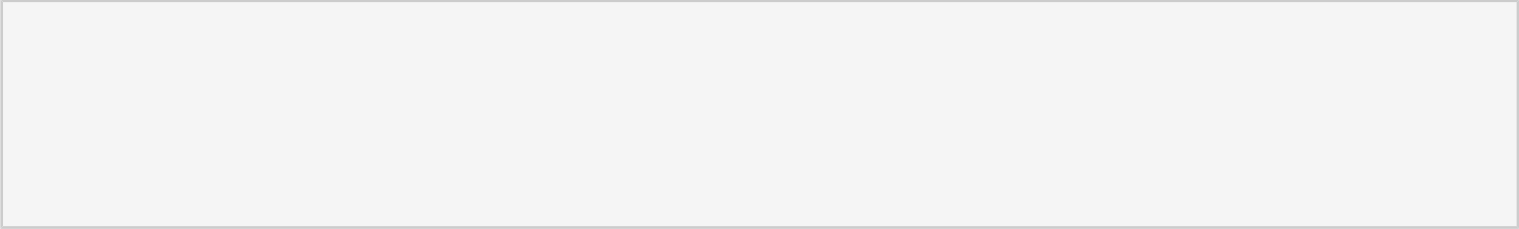
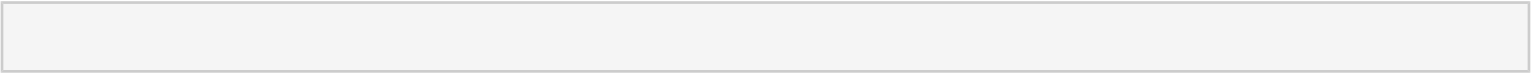
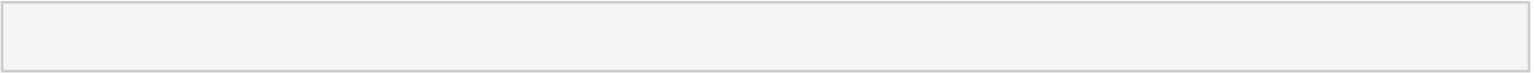





```
0017 % Make window struct
001/ w.name      = 'HFT196D';
```

[Home](#) >



```
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;
0021oe/otu8tum(v)-0.0065;0021oe/otu8tum(v.*v)-0.0065;0in1oe/otu8v-0.0
```

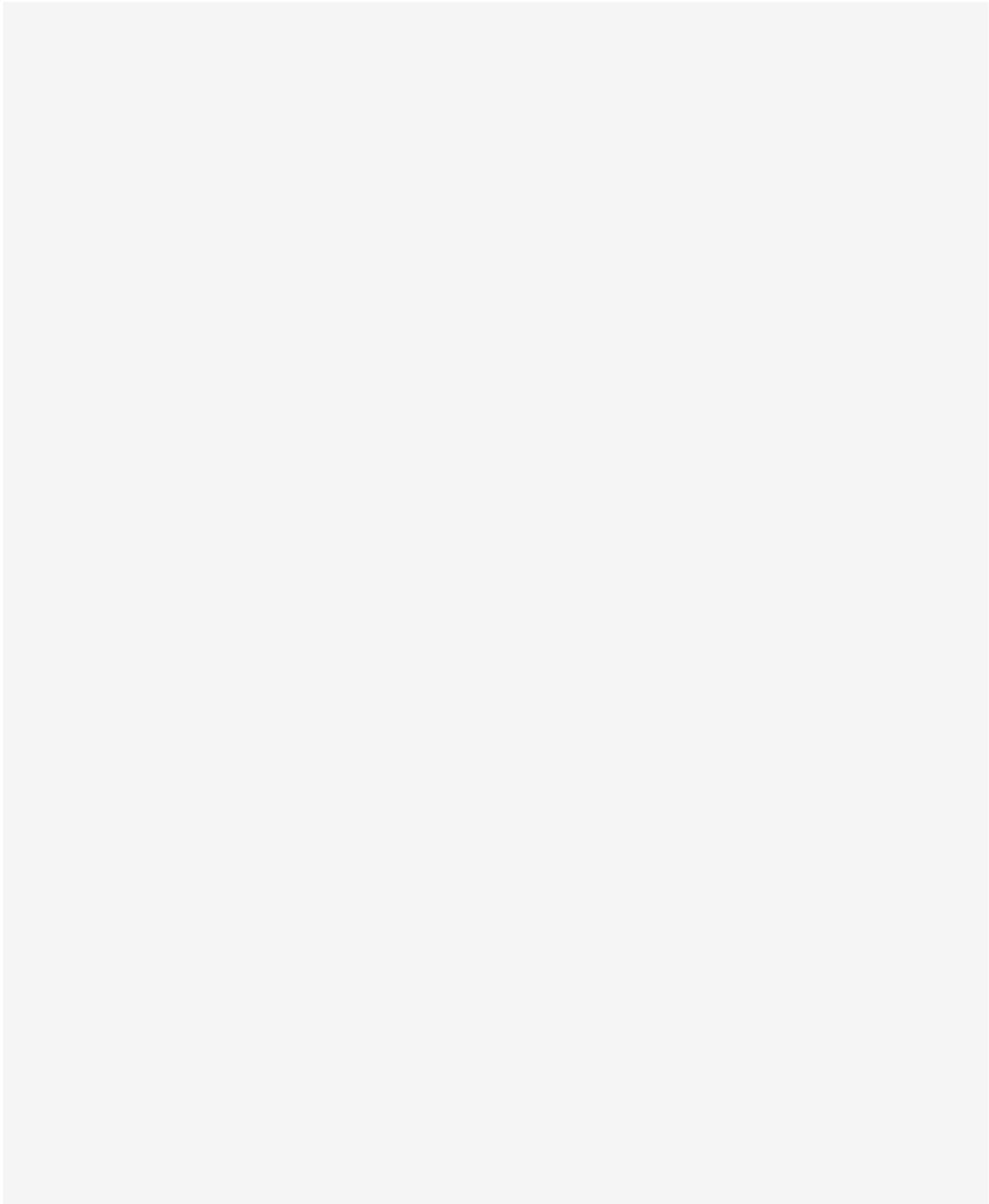

--

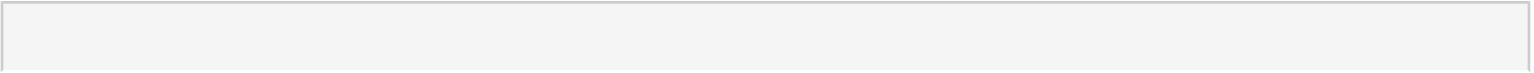
PURPOSE

WIN_HFT95 return HFT95 window sample for input z.

SYNOPSIS








```
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 1ID 0 >>B0 569 569 560.8 rg q 1 0 00 0 5 613.4 cm 0 0 2.5
```



```
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;
```

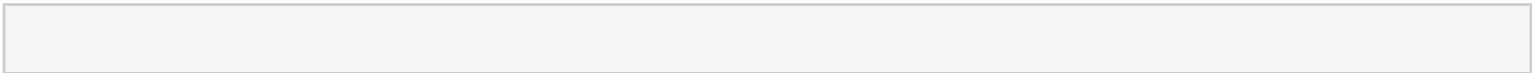
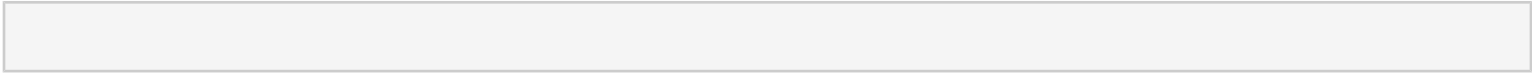
ltpda_launcher_launch_pzmodel

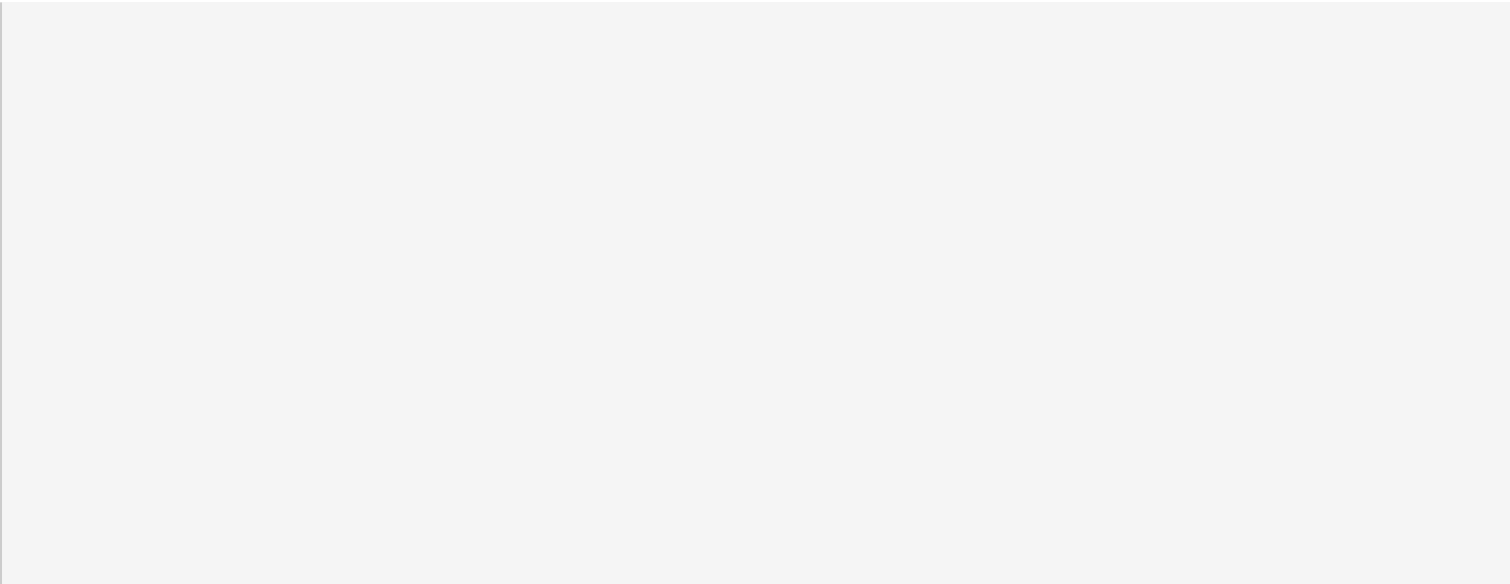
PURPOSE

Callback executed when the LTPDA REPO GUI button is pressed

SYNOPSIS


```
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
```




```
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
```


0024

0025 % END

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003

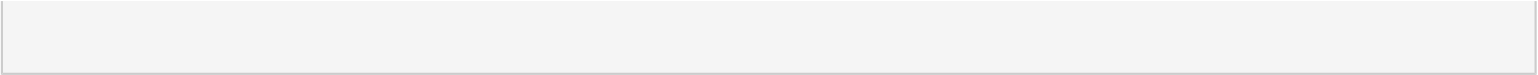


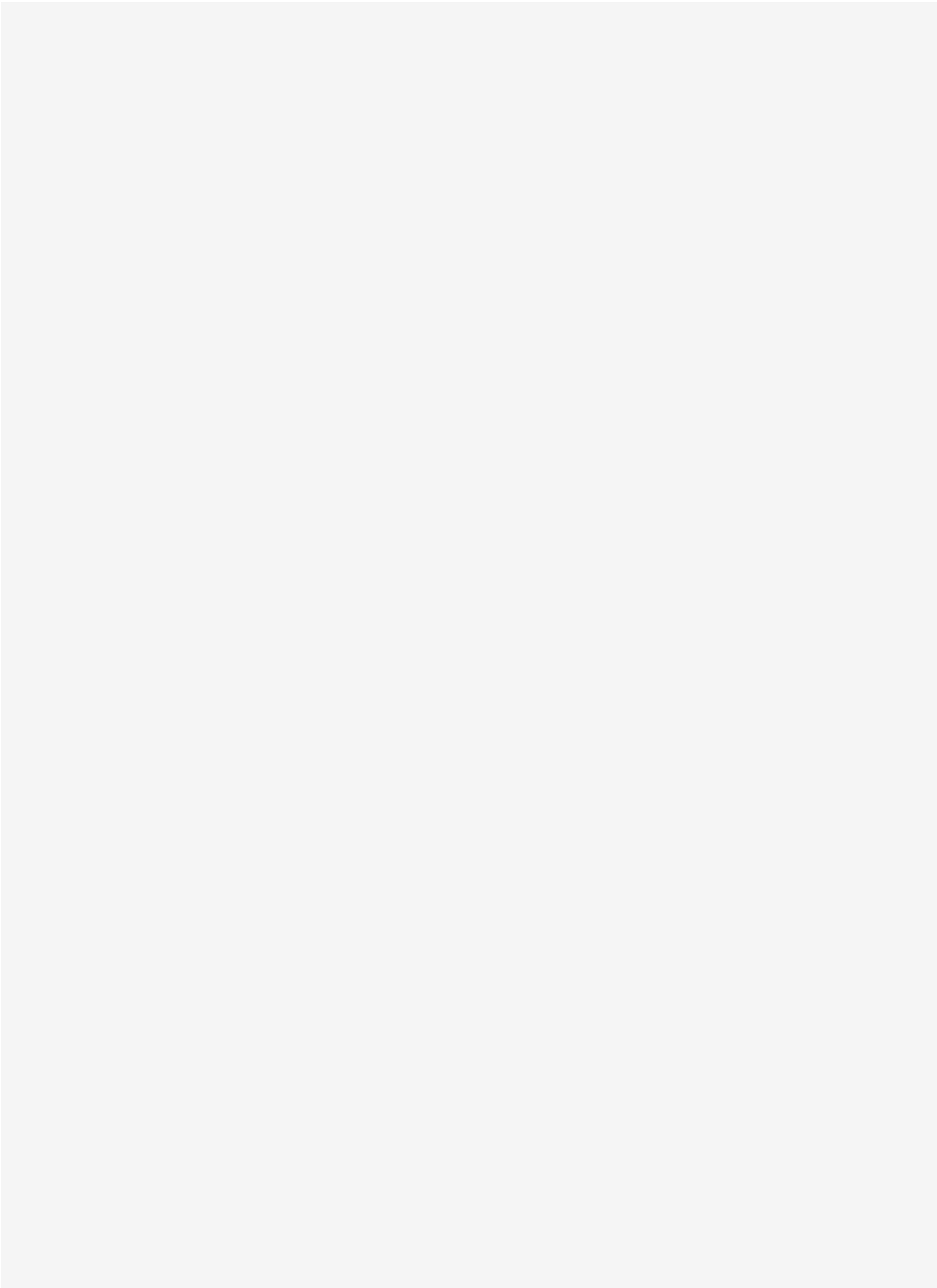
0024

0025 % END

Generated on Tue 25-Mar-2008 23:00:00 by [m2html](#) © 2003


```
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 17:4/2es24mVitson Exp $
0016 %
0017
```



0107

retrieve

PURPOSE

RETRIEVE specwin objects with given ids from the LTPDA repository specified by the input

SYNOPSIS

function objs = retrieve(conn, ids)

DESCRIPTION

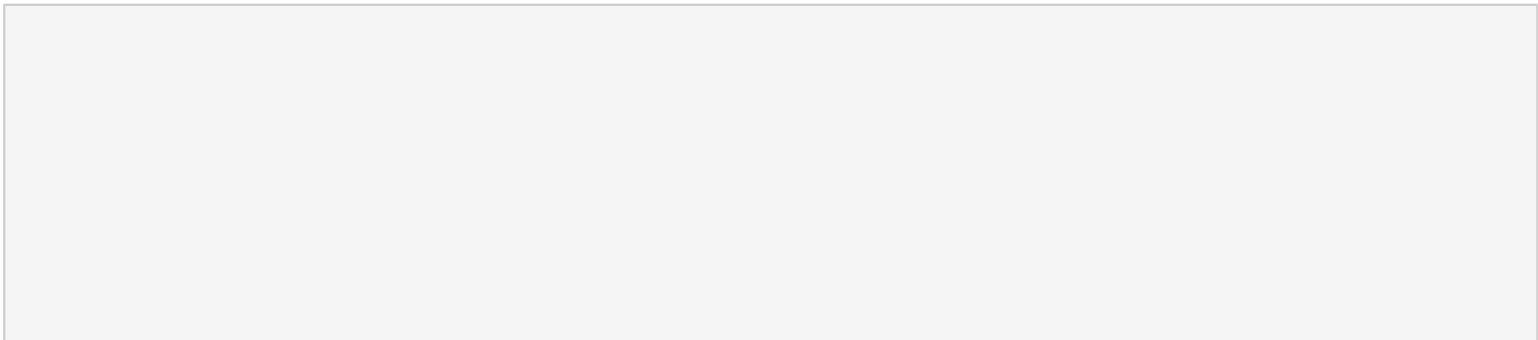
```
RETRIEVE specwin objects with given ids from the LTPDA repository specified by the
input
datab/se connection.

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

DESCRIPTION: RETRIEVE specwin objects with given ids from the LTPDA
             repository specified by the input datab/se connection.

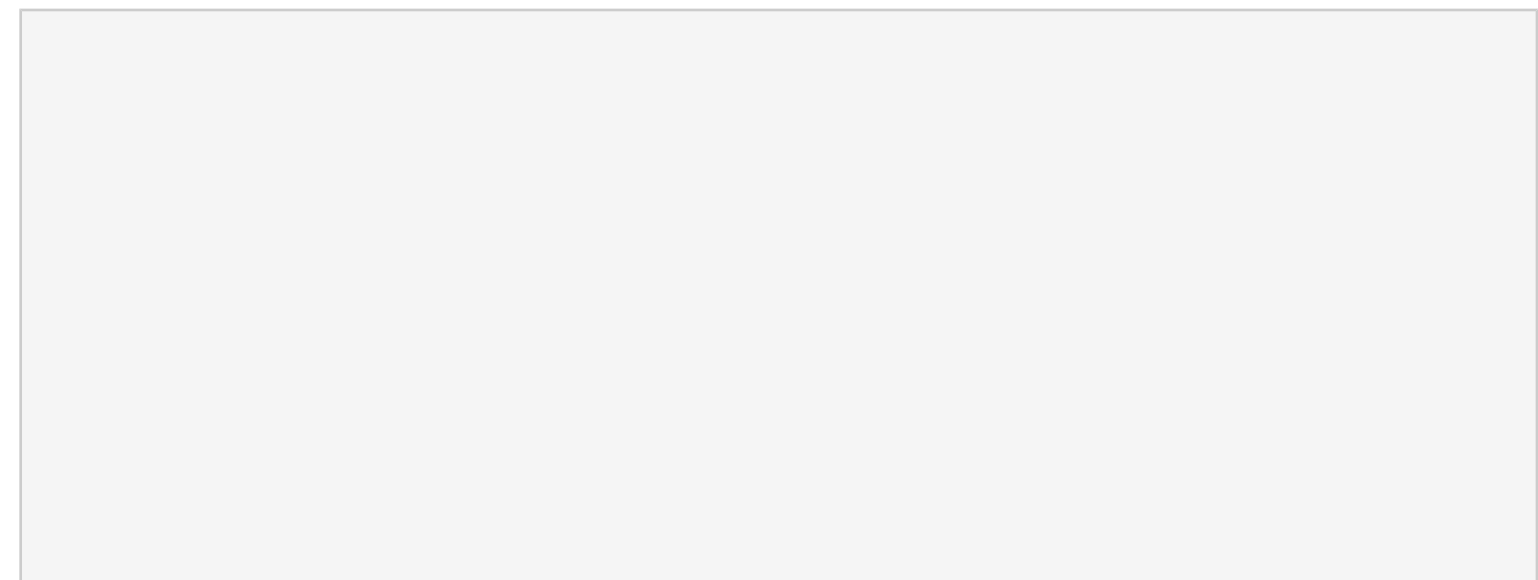
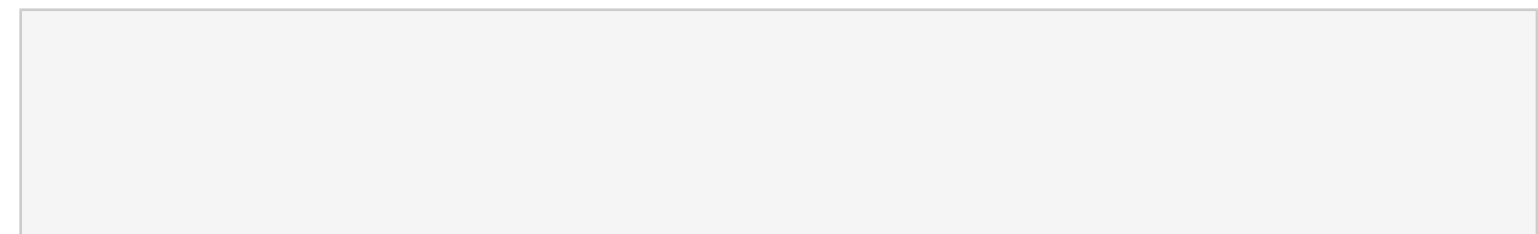
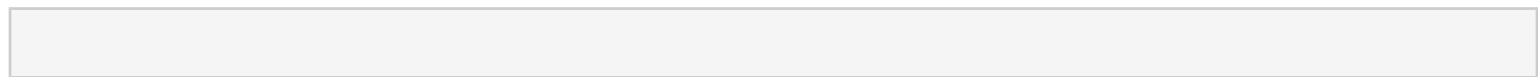
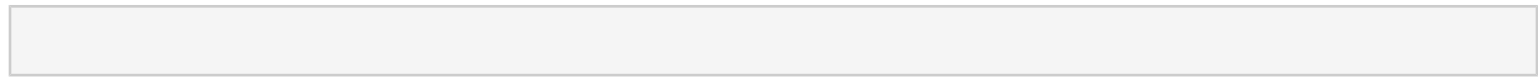
CALL:       p = retrieve(conn, ids)

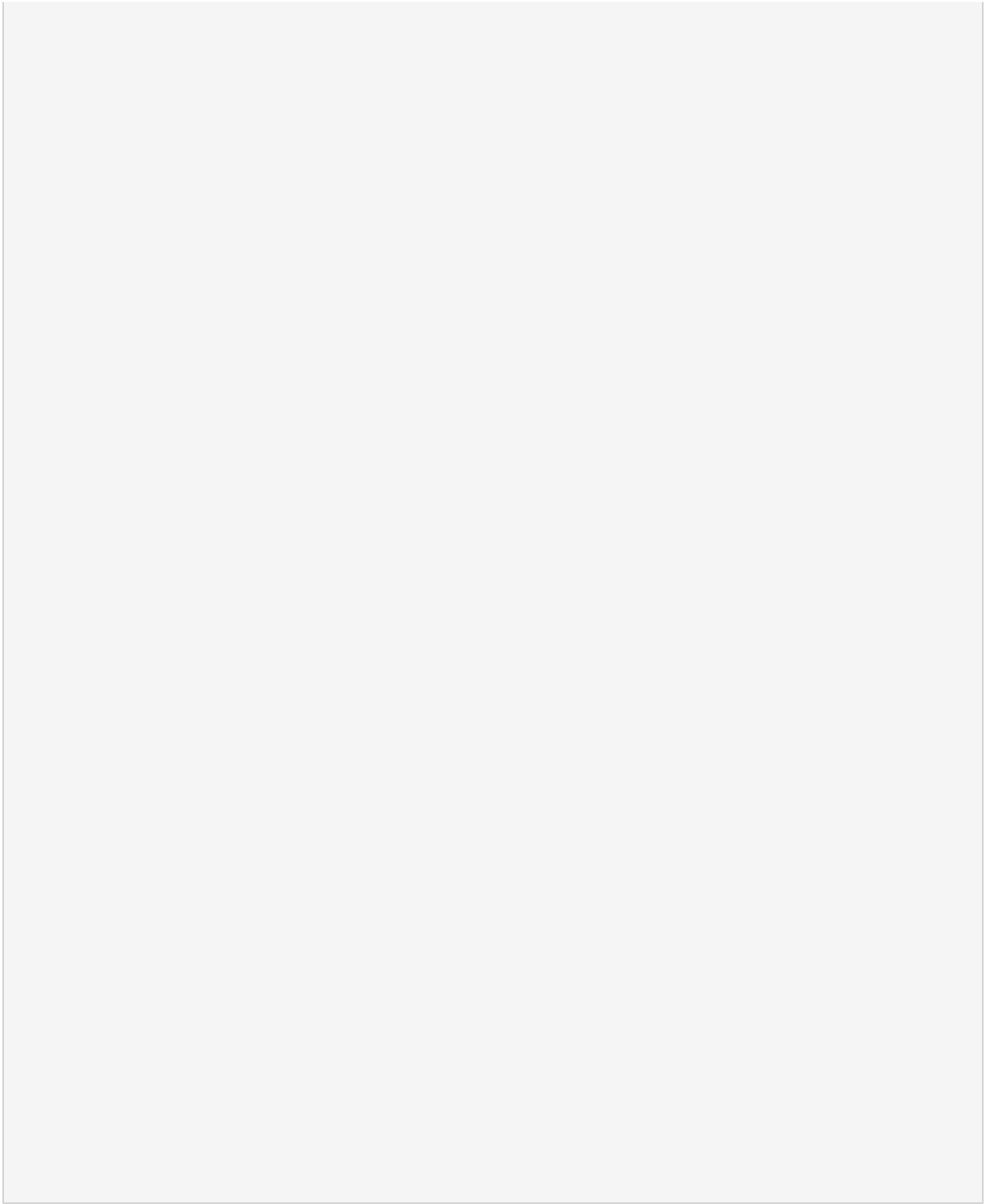
VERSION:    $Id: retrieve.m,v 1.1 2007/08/31 17:40:08 hewitson Exp $
```




```
0009 % repository specified by the input database connection.  
0010 %  
0011 % CALL: p = retrieve(conn, ids)  
0012 %  
0013 %  
0014 %  
0015
```

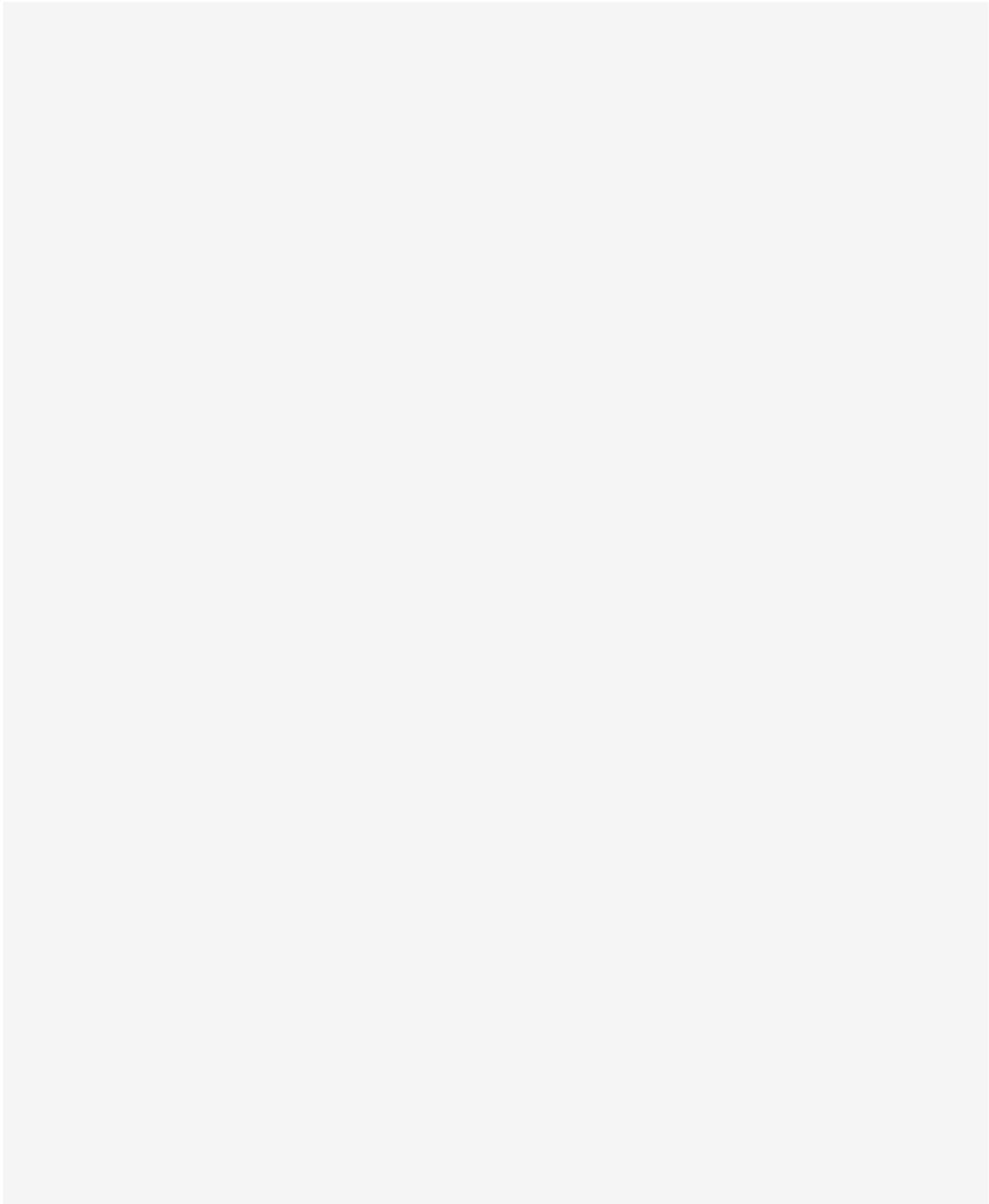
[Home](#) > [S3S3SRB1838I3](#) [EMC](#) /Div <</MCID 1 >>BDC BT 0 0 0 rg 100.974 Tf 14 0 0 1



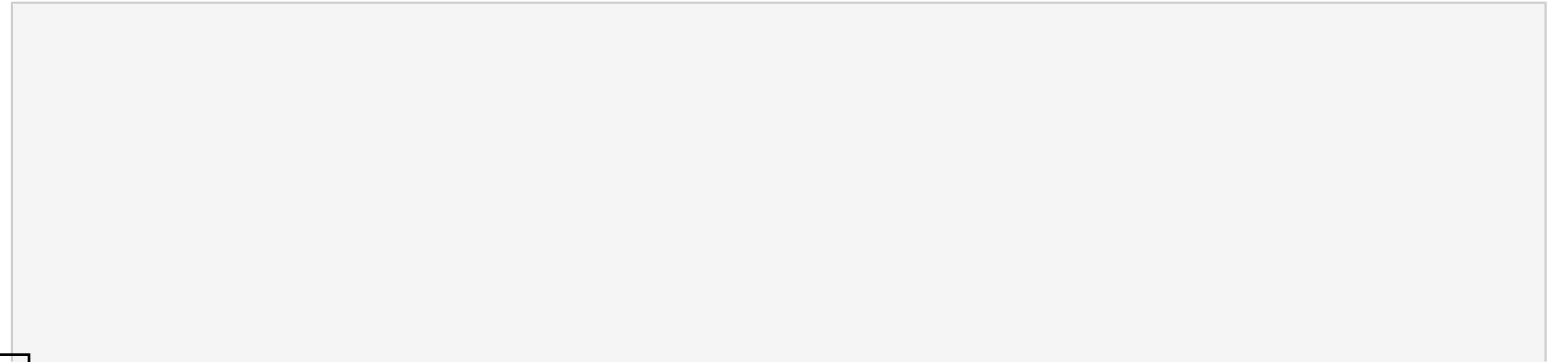
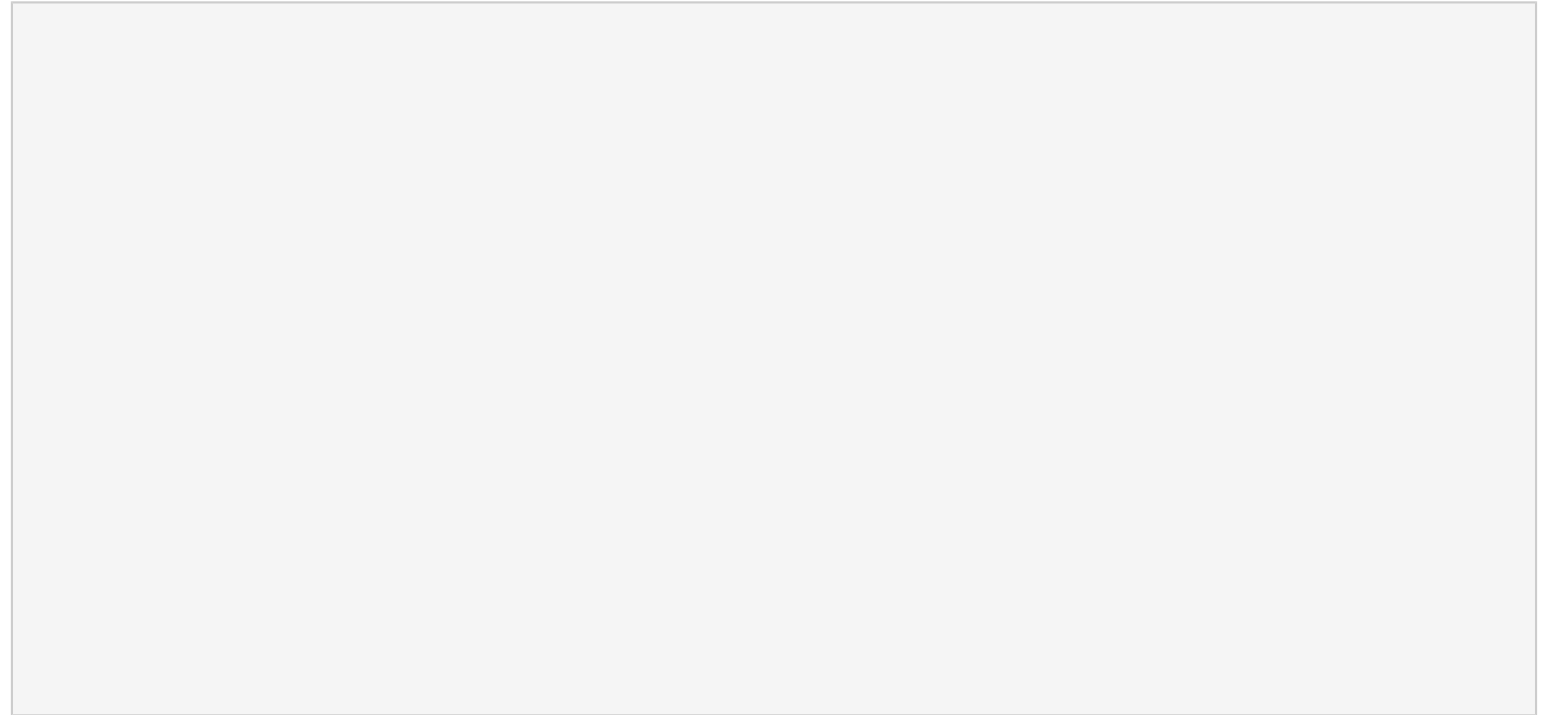
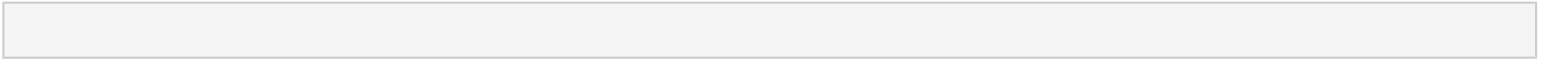
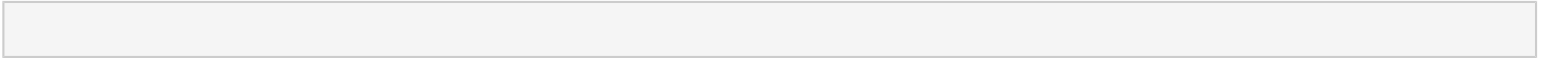


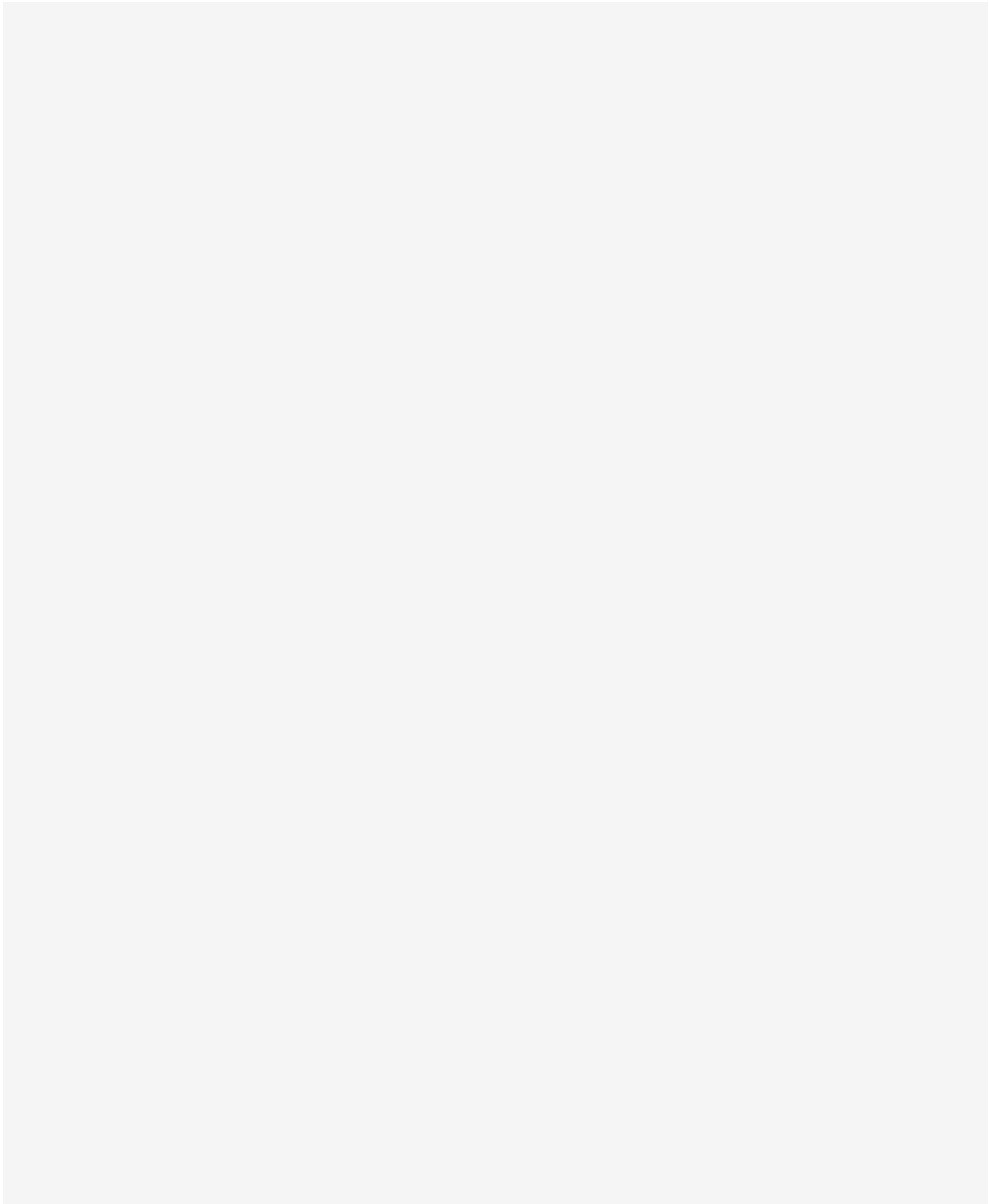
0025 % END





[Home](#) >

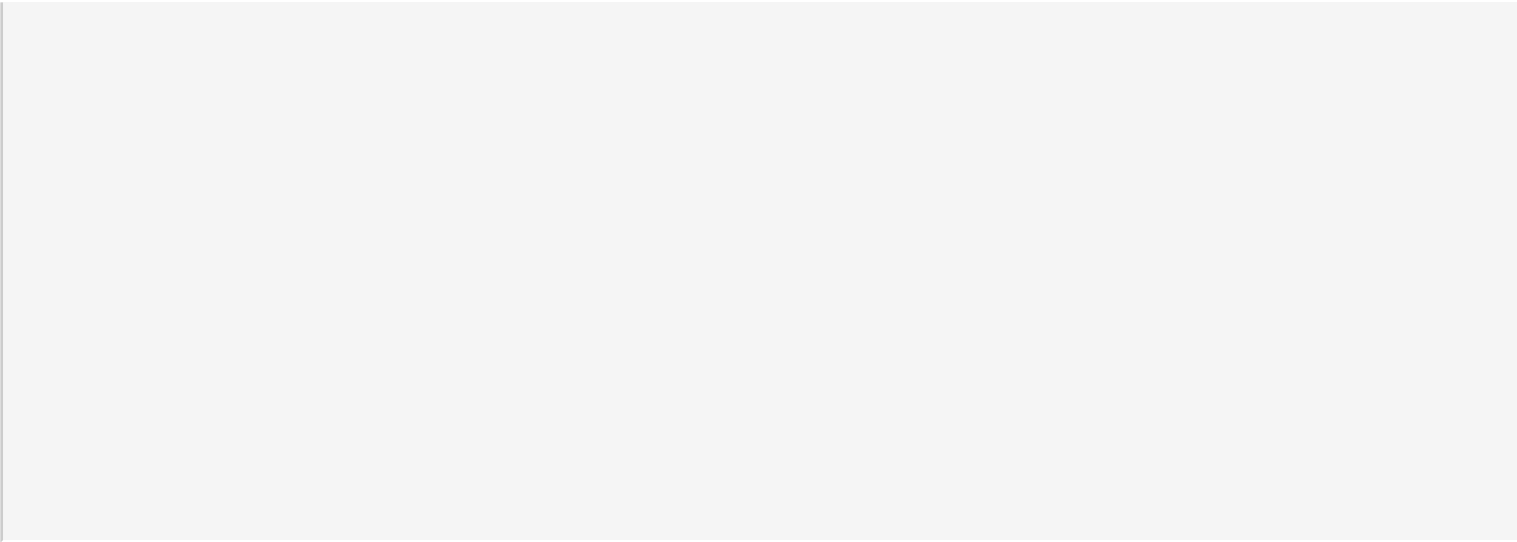


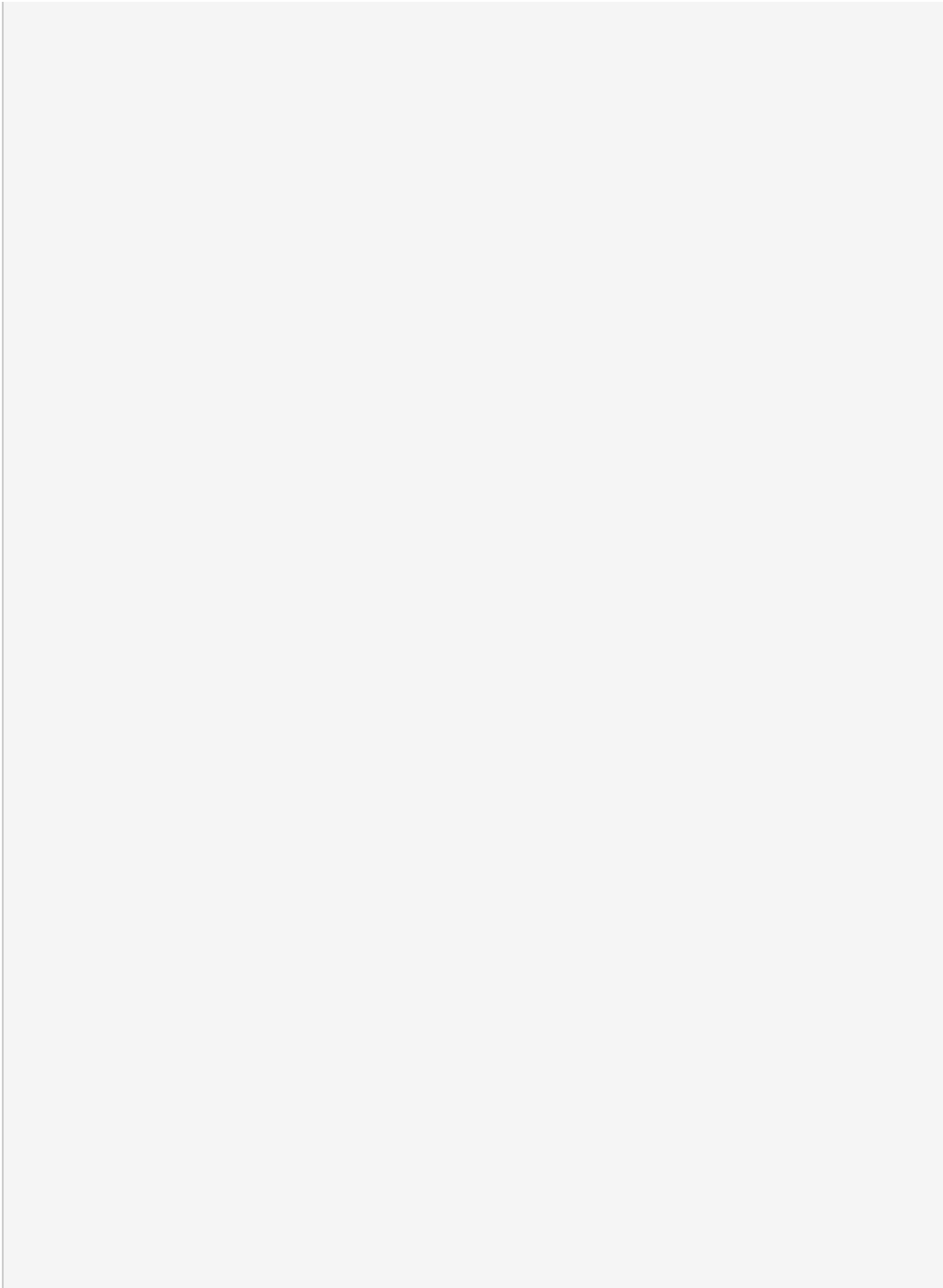



```
0012 % VERSION:      $Id: ltpda_mat2str.m,v 1.5 2008/02/19 17:28:38 ingo Exp $
0013 %
0014 % HISTORY:       26-07-2007 Diepholz
0015 %               Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 MAX_PRECISION = 20;
0020
0021 s = size(number);
0022
```

--

```
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 17:4/2es24mVitson Exp $
0016 %
0017
```




```

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 A0gin{j}, for j=1:Na

0082 a = as(j);

0083 d = a.data;

0084

0085 if isa(d, 'fsdata')

0086

0087 nf =

0088

0089 nfest(d.y, bw, hc);

0088

0089 else

0090 error('### I can only find lines in frequency-series A0g.');

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

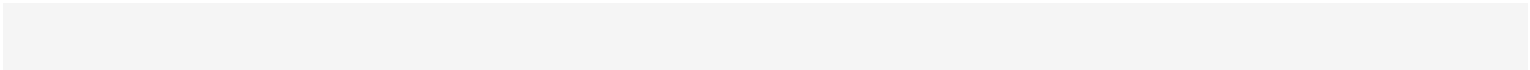
0106 % set name

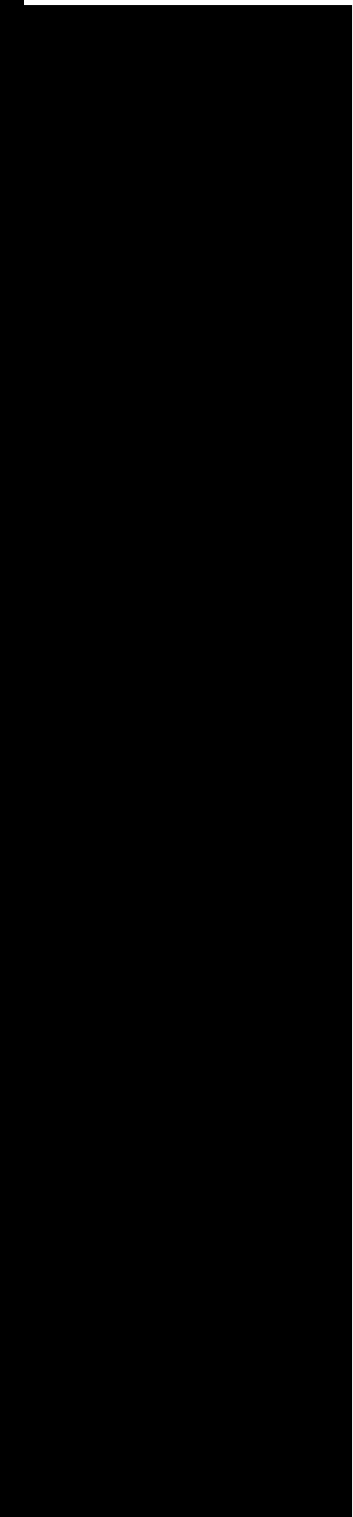
0107 % name for this object

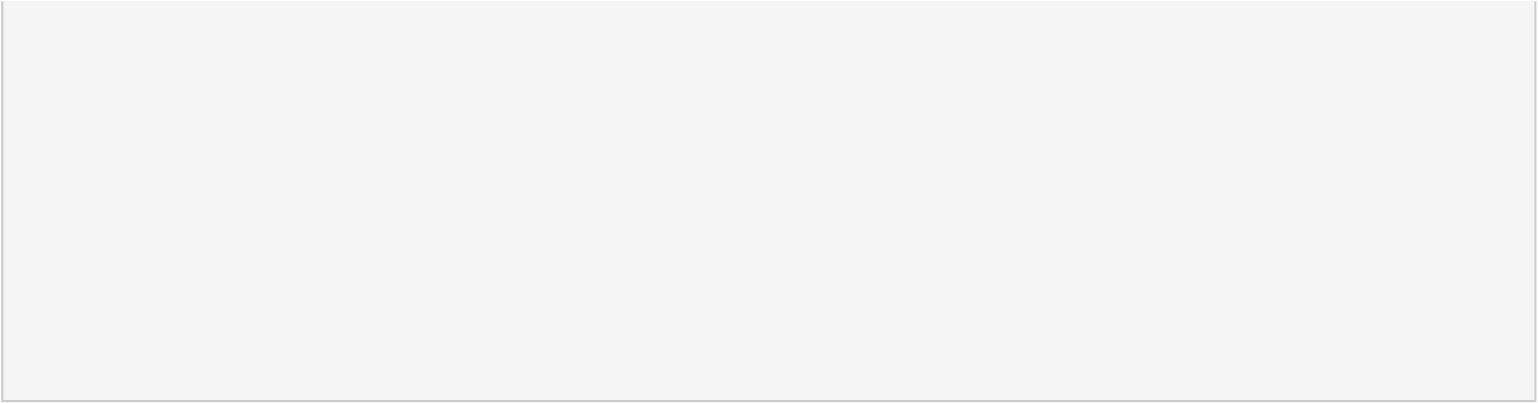
0108 if isempty(invars{j})

```
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
```









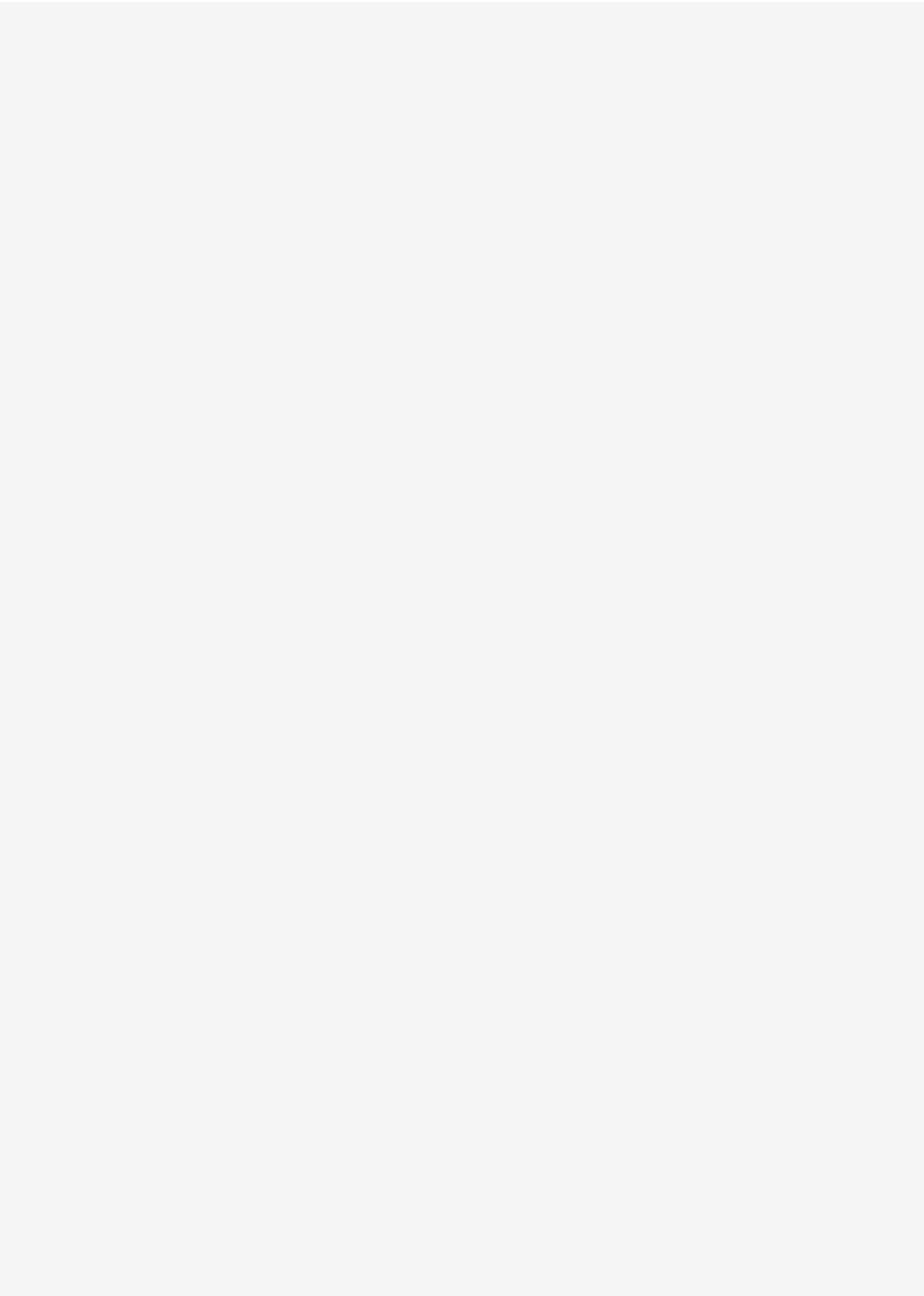

```
0039 %% Process inputs
0040
0041 objs = {};
0042 sinfo = [];
0043 for f4 = 1:54
    for f5 = 1:54
        f6 = [];
        for j = 1:660
            error(.133 * 0.698 - else rg
```


0155 parent = xml.getDocumentElement;

```

0212         disp(sprintf(' + made meta--ata entry'));
0213     catch
0214         error('### failed to make meta--ata entry');
0215     end
0216
0217     %%%%%%%%%%% Update object tables
0218     eid = update_object_tables(conn, obj, objid);
0219     %%%%%%%%%%% Update transactions table
0220
0221     try
0222         message = ltpda_insert(conn, 'transactions',...
0223             'obj_id', objid,...
0224             'user_id', userid,...
0225             'transdate', tdate,...
0226             'direction', 'in'...
0227         );
0228         disp(sprintf(' + updated transactions table'));
0229     catch
0230         error('### Failed to update transactions table');
0231     end
0232
0233     % Collect this ID
0234     ids = [ids objid];
0235 else
0236     warning('!! Object is not one of the User classes. Not submitting. ');
0237     disp(obj);
0238 end
0239 end
0240
0241 %% Now make collection entry
0242 if ~isempty(ids)
0243     try
0244         message = ltpda_insert(conn, 'collections',...
0245             'nobjs', length(ids),...
0229         0230             error(
0238
0242
0229         022             error(02 - onea (02 ('. Not submitting. '))Tj 0 0 0 r56 -1rethrow
022             error(

```






```
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
```


[!\[\]\(529949c2c3dadbaa4e538e8c643454bc_img.jpg\) Master indeL](#)

[IndeL for classes/@cdata/private!\[\]\(3dfb8d66e81160ad61421a3452093d1b_img.jpg\)](#)



[< Master index](#)

Index for classes/@provenance/private

Matlab files directory:

Contents	%% class: provenance/private %%%%%%%%%%
retrieve	RETRIEVE provenance objects with given ids from the LTPDA repository specified by the input

CVS

[m2html](#)



[< Master index](#)

[Index for classes/@xydata/private >](#)

Index for classes/@xydata/private

Matlab files in this directory:

[Contents](#)



Index for classes/@plist/private

Matlab files in this directory:

[Contents](#) %%%%%%%%%%%%%%% class: plist/private %%%%%%%%%%%%%%%
%%%%%%%%%%

[retrieve](#) RETRIEVEte %%% objects with given ids from the LTPDA repository specified by the
input

Subsequent directories:

o CVS





[!\[\]\(1d3a1175dd4902218e694b9c098adb83_img.jpg\) Master index](#)

[Index for classes/@ao/private!\[\]\(c507f772dba2b921f86777f01218e570_img.jpg\)](#)

Index for classes/@ao/private (**>**)Tj ET EM2 /H1 <=





 [utp_ltpda_ss_kalman_filter](#)

tests ltpda_simulate_discrete which simulates a discrete state space system