

API Documentation

May 2, 2007

Contents

Contents	1
1 Package asap	3
1.1 Modules	3
1.2 Functions	3
1.3 Variables	4
2 Module asap.asapfit	6
2.1 Class asapfit	6
2.1.1 Methods	6
3 Module asap.asapfitter	7
3.1 Class fitter	7
3.1.1 Methods	7
4 Module asap.asaplinefind	11
4.1 Class linefinder	11
4.1.1 Methods	11
5 Module asap.asaplot	14
5.1 Class asaplot	14
5.1.1 Methods	14
6 Module asap.asaplotbase	15
6.1 Class asaplotbase	15
6.1.1 Methods	15
7 Module asap.asaplotgui	20
7.1 Class asaplotgui	20
7.1.1 Methods	20
8 Module asap.asaplotgui_gtk	23
8.1 Class asaplotgui	23
8.1.1 Methods	23
9 Module asap.asapmath	26
9.1 Functions	26

10 Module asap.asaplotter	28
10.1 Class asaplotter	28
10.1.1 Methods	28
11 Module asap.asapreader	38
11.1 Class reader	38
11.1.1 Methods	38
12 Module asap.linecatalog	40
12.1 Variables	40
12.2 Class linecatalog	40
12.2.1 Methods	40
13 Module asap.scantable	42
13.1 Class scantable	42
13.1.1 Methods	42
14 Module asap.selector	60
14.1 Class selector	60
14.1.1 Methods	60
Index	64

1 Package asap

This is the ATNF Single Dish Analysis package.

1.1 Modules

- **asapfit** (Section 2, p. 6)
- **asapfitter** (Section 3, p. 7)
- **asaplinefind** (Section 4, p. 11)
- **asaplot**: ASAP plotting class based on matplotlib. (Section 5, p. 14)
- **asaplotbase**: ASAP plotting class based on matplotlib. (Section 6, p. 15)
- **asaplotgui**: ASAP plotting class based on matplotlib. (Section 7, p. 20)
- **asaplotgui_gtk**: ASAP plotting class based on matplotlib. (Section 8, p. 23)
- **asapmath** (Section 9, p. 26)
- **asapplotter** (Section 10, p. 28)
- **asapreader** (Section 11, p. 38)
- **linecatalog**: A representation of a spectra line catalog. (Section 12, p. 40)
- **scantable** (Section 13, p. 42)
- **selector** (Section 14, p. 60)

1.2 Functions

```
is_ipython()
```

```
list_files(path='.', suffix='rpf')
```

Return a list files readable by asap, such as rpf, sdfits, mbf, asap

Parameters:

path: The directory to list (default '.')
 suffix: The file extension (default rpf)

Example:

```
files = list_files("data/","sdfits")
print files
['data/2001-09-01_0332_P363.sdfits',
 'data/2003-04-04_131152_t0002.sdfits',
 'data/Sgr_86p262_best_SPC.sdfits']
```

```
list_rcparameters()
```

```
mask_and(a, b)
```

```
mask_not(a)
```

```
mask_or(a, b)
```

```
print_log()
```

```
rc(group, **kwargs)
```

Set the current rc params. Group is the grouping for the rc, eg for scantable.save the group is 'scantable', for plotter.stacking, the group is 'plotter', and so on. kwargs is a list of attribute name/value pairs, eg

```
rc('scantable', save='SDFITS')
```

sets the current rc params and is equivalent to

```
rcParams['scantable.save'] = 'SDFITS'
```

Use rcdefaults to restore the default rc params after changes.

```
rc_params()
```

Return the default params updated from the values in the rc file

```
rcdefaults()
```

Restore the default rc params - the ones that were created at asap load time

```
unique(x)
```

Return the unique values in a list

Parameters:

```
x:      the list to reduce
```

Examples:

```
x = [1,2,3,3,4]
```

```
print unique(x)
```

```
[1,2,3,4]
```

```
welcome()
```

1.3 Variables

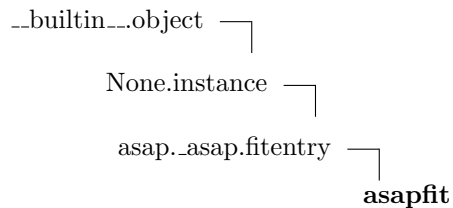
Name	Description
__date__	Value: '2007-05-02' (<i>type=str</i>)
__version__	Value: '2.2.0' (<i>type=str</i>)

continued on next page

Name	Description
asaplog	Value: <asap._asap.Log object at 0x40206504> (<i>type=Log</i>)
defaultParams	Value: {'scantable.verbosesummary': [False, <function -_validate_bool at 0x401fc4fc>]... (<i>type=dict</i>)
plf	Value: 'linux.gnu' (<i>type=str</i>)
rcParams	Value: {'scantable.verbosesummary': False, 'scantable.-autoaverage': True, 'verbose':... (<i>type=dict</i>)

2 Module *asap.asapfit*

2.1 Class *asapfit*



2.1.1 Methods

<code>__init__(self, other=None)</code> Overrides: <code>asap._asap.fitentry.__init__</code>

<code>__str__(self)</code> Overrides: <code>__builtin__.object.__str__</code>
--

<code>as_dict(self)</code>

<code>save(self, filename)</code>

Inherited from instance: `__new__`

Inherited from object: `__delattr__`, `__getattr__`, `__hash__`, `__reduce__`, `__reduce_ex__`, `__repr__`, `__setattr__`

3 Module *asap.asapfitter*

3.1 Class *fitter*

The fitting class for ASAP.

3.1.1 Methods

`__init__(self)`

Create a fitter object. No state is set.

`auto_fit(self, insitu=None, plot=False)`

Return a scan where the function is applied to all rows for all Beams/IFs/Pols.

`commit(self)`

Return a new scan where the fits have been committed (subtracted)

`fit(self, row=0, estimate=False)`

Execute the actual fitting process. All the state has to be set.

Parameters:

row: specify the row in the scantable
estimate: auto-compute an initial parameter set (default False)
This can be used to compute estimates even if fit was called before.

Example:

```
s = scantable('myscan.asap')
s.set_cursor(thepol=1)      # select second pol
f = fitter()
f.set_scan(s)
f.set_function(poly=0)
f.fit(row=0)                # fit first row
```

`get_area(self, component=None)`

Return the area under the fitted gaussian component.

Parameters:

component: the gaussian component selection,
default (None) is the sum of all components

Note:

This will only work for gaussian fits.

get_chi2(*self*)Return χ^2 .**get_errors**(*self*, *component=None*)

Return the errors in the parameters.

Parameters:

component: get the errors for the specified component only, default is all components

get_estimate(*self*)

Return the parameter estimates (for non-linear functions).

get_fit(*self*)

Return the fitted ordinate values.

get_parameters(*self*, *component=None*, *errors=False*)

Return the fit parameters.

Parameters:

component: get the parameters for the specified component only, default is all components

get_residual(*self*)

Return the residual of the fit.

plot(*self*, *residual=False*, *components=None*, *plotparms=False*, *filename=None*)

Plot the last fit.

Parameters:

residual: an optional parameter indicating if the residual should be plotted (default 'False')

components: a list of components to plot, e.g [0,1], -1 plots the total fit. Default is to only plot the total fit.

plotparms: Indicates if the parameter values should be present on the plot

set_data(*self*, *xdat*, *ydat*, *mask*=None)

Set the abscissa and ordinate for the fit. Also set the mask indicating valid points.
 This can be used for data vectors retrieved from a scantable.
 For scantable fitting use 'fitter.set_scan(scan, mask)'.

Parameters:

xdat: the abscissa values
ydat: the ordinate values
mask: an optional mask

set_function(*self*, ****kwargs**)

Set the function to be fit.

Parameters:

poly: use a polynomial of the order given
gauss: fit the number of gaussian specified

Example:

fitter.set_function(gauss=2) # will fit two gaussians
 fitter.set_function(poly=3) # will fit a 3rd order polynomial

set_gauss_parameters(*self*, *peak*, *centre*, *fwhm*, *peakfixed*=0, *centerfixed*=0, *fwhmfixed*=0, *component*=0)

Set the Parameters of a 'Gaussian' component, set with set_function.

Parameters:

peak, *centre*, *fwhm*: The gaussian parameters
peakfixed,
centerfixed,
fwhmfixed: Optional parameters to indicate if the parameters should be held fixed during the fitting process. The default is to keep all parameters flexible.
component: The number of the component (Default is the component 0)

set_parameters(*self*, **args*, ****kwargs**)

Set the parameters to be fitted.

Parameters:

params: a vector of parameters
fixed: a vector of which parameters are to be held fixed (default is none)
component: in case of multiple gaussians, the index of the component

set_scan(*self*, *thescan*=None, *mask*=None)

Set the 'data' (a scantable) of the fitter.

Parameters:

thescan: a scantable
 mask: a msk retireved from the scantable

store_fit(*self*, *filename*=None)

Save the fit parameters.

Parameters:

filename: if specified save as an ASCII file, if None (default)
 store it in the scnatable

4 Module *asap.asaplinefind*

4.1 Class *linefinder*

The class for automated spectral line search in ASAP.

Example:

```
fl=linefinder()
fl.set_scan(sc)
fl.set_options(threshold=3)
nlines=fl.find_lines(edge=(50,0))
if nlines!=0:
    print "Found ",nlines," spectral lines"
    print fl.get_ranges(False)
else:
    print "No lines found!"
sc2=sc.poly_baseline(fl.get_mask(),7)
```

The algorithm involves a simple threshold criterion. The line is considered to be detected if a specified number of consecutive channels (default is 3) is brighter (with respect to the current baseline estimate) than the threshold times the noise level. This criterion is applied in the iterative procedure updating baseline estimate and trying reduced spectral resolutions to detect broad lines as well. The off-line noise level is determined at each iteration as an average of 80% of the lowest variances across the spectrum (i.e. histogram equalization is used to avoid missing weak lines if strong ones are present). For bad baseline shapes it is recommended to increase the threshold and possibly switch the averaging option off (see `set_options`) to detect strong lines only, fit a high order baseline and repeat the line search.

4.1.1 Methods

<code>__init__(self)</code>
Create a line finder object.

find_lines(*self*, *nRow*=0, *mask*=[], *edge*=(0, 0))

Search for spectral lines in the scan assigned in `set_scan`.

Parameters:

`nRow`: a row in the scantable to work with
`mask`: an optional mask (e.g. retrieved from scantable)
`edge`: an optional number of channels to drop at the edge of the spectrum. If only one value is specified, the same number will be dropped from both sides of the spectrum. Default is to keep all channels

A number of lines found will be returned

get_mask(*self*, *invert*=False)

Get the mask to mask out all lines that have been found (default)

Parameters:

`invert` if True, only channels belong to lines will be unmasked

Note: all channels originally masked by the input mask or dropped out by the edge parameter will still be excluded regardless on the invert option

get_ranges(*self*, *defunits*=True)

Get ranges (start and end channels or velocities) for all spectral lines found.

Parameters:

`defunits` if True (default), the range will use the same units as set for the scan (e.g. LSR velocity)
if False, the range will be expressed in channels

```
set_options(self, threshold=1.7320508075688772, min_nchan=3, avg_limit=8,  
box_size=0.20000000000000001)
```

Set the parameters of the algorithm

Parameters:

<code>threshold</code>	a single channel S/N ratio above which the channel is considered to be a detection Default is $\sqrt{3}$, which together with <code>min_nchan=3</code> gives a 3-sigma criterion
<code>min_nchan</code>	a minimal number of consecutive channels, which should satisfy a threshold criterion to be a detection. Default is 3.
<code>avg_limit</code>	A number of consecutive channels not greater than this parameter can be averaged to search for broad lines. Default is 8.
<code>box_size</code>	A running mean box size specified as a fraction of the total spectrum length. Default is 1/5

Note: For bad baselines `threshold` should be increased, and `avg_limit` decreased (or even switched off completely by setting this parameter to 1) to avoid detecting baseline undulations instead of real lines.

```
set_scan(self, scan)
```

Set the 'data' (scantable) to work with.

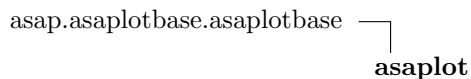
Parameters:

<code>scan:</code>	a scantable
--------------------	-------------

5 Module *asap.asaplot*

ASAP plotting class based on matplotlib.

5.1 Class *asaplot*



ASAP plotting class based on matplotlib.

5.1.1 Methods

<code>__init__(self, rows=1, cols=0, title='', size=(8, 4), buffering=False)</code>
Create a new instance of the ASAPlot plotting class. If rows < 1 then a separate call to <code>set_panels()</code> is required to define the panel layout; refer to the doctext for <code>set_panels()</code> .
Overrides: <code>asap.asaplotbase.asaplotbase.__init__</code>

Inherited from *asaplotbase*: `clear`, `delete`, `get_line`, `hist`, `hold`, `legend`, `palette`, `plot`, `position`, `region`, `register`, `release`, `save`, `set_axes`, `set_figure`, `set_limits`, `set_line`, `set_panels`, `set_title`, `show`, `subplot`, `text`, `tidy`, `vline_with_label`

6 Module *asap.asaplotbase*

ASAP plotting class based on matplotlib.

6.1 Class *asaplotbase*

Known Subclasses: *asaplot*, *asaplotgui*, *asaplotgui*

ASAP plotting base class based on matplotlib.

6.1.1 Methods

`__init__(self, rows=1, cols=0, title='', size=(8, 6), buffering=False)`

Create a new instance of the ASAPPlot plotting class.

If rows < 1 then a separate call to `set_panels()` is required to define the panel layout; refer to the doctext for `set_panels()`.

`clear(self)`

Delete all lines from the plot. Line numbering will restart from 0.

`delete(self, numbers=None)`

Delete the 0-relative line number, default is to delete the last. The remaining lines are NOT renumbered.

`get_line(self)`

Get the current default line attributes.

`hist(self, x=None, y=None, fmt=None, add=None)`

Plot a histogram. N.B. the x values refer to the start of the histogram bin.
fmt is the line style as in `plot()`.

`hold(self, hold=True)`

Buffer graphics until subsequently released.

legend(*self*, *loc*=None)

Add a legend to the plot.

Any other value for *loc* else disables the legend:

- 1: upper right
- 2: upper left
- 3: lower left
- 4: lower right
- 5: right
- 6: center left
- 7: center right
- 8: lower center
- 9: upper center
- 10: center

palette(*self*, *color*, *colormap*=None, *linestyle*=0, *linestyles*=None)

plot(*self*, *x*=None, *y*=None, *fmt*=None, *add*=None)

Plot the next line in the current frame using the current line attributes. The ASAPlot graphics window will be mapped and raised.

The argument list works a bit like the matlab `plot()` function.

position(*self*)

Use the mouse to get a position from a graph.

region(*self*)

Use the mouse to get a rectangular region from a plot.

The return value is `[x0, y0, x1, y1]` in world coordinates.

register(*self*, *type*=None, *func*=None)

Register, reregister, or deregister events of type 'button_press', 'button_release', or 'motion_notify'.

The specified callback function should have the following signature:

```
def func(event)
```

where event is an `MplEvent` instance containing the following data:

```
name           # Event name.
canvas         # FigureCanvas instance generating the event.
x = None      # x position - pixels from left of canvas.
y = None      # y position - pixels from bottom of canvas.
button = None  # Button pressed: None, 1, 2, 3.
key = None    # Key pressed: None, chr(range(255)), shift,
              # win, or control
inaxes = None  # Axes instance if cursor within axes.
xdata = None   # x world coordinate.
ydata = None   # y world coordinate.
```

For example:

```
def mouse_move(event):
    print event.xdata, event.ydata

a = asaplot()
a.register('motion_notify', mouse_move)
```

If `func` is `None`, the event is deregistered.

Note that in TkAgg keyboard button presses don't generate an event.

release(*self*)

Release buffered graphics.

save(*self*, *fname*=None, *orientation*=None, *dpi*=None, *papertype*=None)

Save the plot to a file.

`fname` is the name of the output file. The image format is determined from the file suffix; 'png', 'ps', and 'eps' are recognized. If no file name is specified 'yyyymmdd_hhmmss.png' is created in the current directory.

set_axes(*self*, *what=None*, **args*, ***kwargs*)

Set attributes for the axes by calling the relevant Axes.set_*(*i*) method. Colour translation is done as described in the doctext for palette().

set_figure(*self*, *what=None*, **args*, ***kwargs*)

Set attributes for the figure by calling the relevant Figure.set_*(*i*) method. Colour translation is done as described in the doctext for palette().

set_limits(*self*, *xlim=None*, *ylim=None*)

Set x-, and y-limits for each subplot.

xlim = [*xmin*, *xmax*] as in axes.set_xlim(). *ylim* = [*ymin*, *ymax*] as in axes.set_ylim().

set_line(*self*, *number=None*, ***kwargs*)

Set attributes for the specified line, or else the next line(s) to be plotted.

number is the 0-relative number of a line that has already been plotted. If no such line exists, attributes are recorded and used for the next line(s) to be plotted.

Keyword arguments specify Line2D attributes, e.g. *color='r'*. Do

```
import matplotlib
help(matplotlib.lines)
```

The set_* methods of class Line2D define the attribute names and values. For non-US usage, "colour" is recognized as synonymous with "color".

Set the value to None to delete an attribute.

Colour translation is done as described in the doctext for palette().

set_panels(*self*, *rows=1*, *cols=0*, *n=-1*, *nplots=-1*, *ganged=True*)

Set the panel layout.

rows and *cols*, if *cols* != 0, specify the number of rows and columns in a regular layout. (Indexing of these panels in matplotlib is row-major, i.e. column varies fastest.)

cols == 0 is interpreted as a rectangular layout that accomodates '*rows*' panels, e.g. *rows* == 6, *cols* == 0 is equivalent to *rows* == 2, *cols* == 3.

$0 \leq n < rows * cols$ is interpreted as the 0-relative panel number in the configuration specified by *rows* and *cols* to be added to the current figure as its next 0-relative panel number (*i*). This allows non-regular panel layouts to be constructed via multiple calls. Any other value of *n* clears the plot and produces a rectangular array of empty panels. The number of these may be limited by *nplots*.

set_title(*self*, *title=None*)

Set the title of the plot window. Use the previous title if title is omitted.

show(*self*, *hardrefresh=True*)

Show graphics dependent on the current buffering state.

subplot(*self*, *i=None*, *inc=None*)

Set the subplot to the 0-relative panel number as defined by one or more invokations of `set_panels()`.

text(*self*, **args*, ***kwargs*)

Add text to the figure.

tidy(*self*)

vline_with_label(*self*, *x*, *y*, *label*, *location='bottom'*, *rotate=0.0*, ***kwargs*)

Plot a vertical line with label. It takes "world" values for x and y.

7 Module *asap.asaplotgui*

ASAP plotting class based on matplotlib.

7.1 Class *asaplotgui*



ASAP plotting class based on matplotlib.

7.1.1 Methods

<code>__init__(self, rows=1, cols=0, title='', size=(8, 6), buffering=False)</code>
Create a new instance of the ASAPPlot plotting class. If rows < 1 then a separate call to <code>set_panels()</code> is required to define the panel layout; refer to the doctext for <code>set_panels()</code> . Overrides: <code>asap.asaplotbase.asaplotbase.__init__</code>
<code>map(self)</code>
Reveal the ASAPPlot graphics window and bring it to the top of the window stack.
<code>position(self)</code>
Use the mouse to get a position from a graph. Overrides: <code>asap.asaplotbase.asaplotbase.position</code>
<code>quit(self)</code>
Destroy the ASAPPlot graphics window.
<code>region(self)</code>
Use the mouse to get a rectangular region from a plot. The return value is <code>[x0, y0, x1, y1]</code> in world coordinates. Overrides: <code>asap.asaplotbase.asaplotbase.region</code>

register(*self*, *type*=None, *func*=None)

Register, reregister, or deregister events of type 'button_press', 'button_release', or 'motion_notify'.

The specified callback function should have the following signature:

```
def func(event)
```

where event is an `MplEvent` instance containing the following data:

```
name           # Event name.
canvas         # FigureCanvas instance generating the event.
x = None       # x position - pixels from left of canvas.
y = None       # y position - pixels from bottom of canvas.
button = None  # Button pressed: None, 1, 2, 3.
key = None     # Key pressed: None, chr(range(255)), shift,
               # win, or control
inaxes = None  # Axes instance if cursor within axes.
xdata = None   # x world coordinate.
ydata = None   # y world coordinate.
```

For example:

```
def mouse_move(event):
    print event.xdata, event.ydata

a = asaplot()
a.register('motion_notify', mouse_move)
```

If `func` is `None`, the event is deregistered.

Note that in TkAgg keyboard button presses don't generate an event.

Overrides: `asap.asaplotbase.asaplotbase.register`

show(*self*, *hardrefresh*=True)

Show graphics dependent on the current buffering state.

Overrides: `asap.asaplotbase.asaplotbase.show`

terminate(*self*)

Clear the figure.

unmap(*self*)

Hide the ASAPlot graphics window.

Inherited from `asaplotbase`: `clear`, `delete`, `get_line`, `hist`, `hold`, `legend`, `palette`, `plot`, `release`, `save`, `set_axes`,

set_figure, set_limits, set_line, set_panels, set_title, subplot, text, tidy, vline_with_label

8 Module `asap.asaplotgui_gtk`

ASAP plotting class based on matplotlib.

8.1 Class `asaplotgui`



ASAP plotting class based on matplotlib.

8.1.1 Methods

<code>__init__(self, rows=1, cols=0, title='', size=(8, 6), buffering=False)</code>
Create a new instance of the ASAPPlot plotting class. If <code>rows < 1</code> then a separate call to <code>set_panels()</code> is required to define the panel layout; refer to the doctext for <code>set_panels()</code> . Overrides: <code>asap.asaplotbase.asaplotbase.__init__</code>
<code>map(self)</code>
Reveal the ASAPPlot graphics window and bring it to the top of the window stack.
<code>position(self)</code>
Use the mouse to get a position from a graph. Overrides: <code>asap.asaplotbase.asaplotbase.position</code>
<code>quit(self)</code>
Destroy the ASAPPlot graphics window.
<code>region(self)</code>
Use the mouse to get a rectangular region from a plot. The return value is <code>[x0, y0, x1, y1]</code> in world coordinates. Overrides: <code>asap.asaplotbase.asaplotbase.region</code>

register(*self*, *type*=None, *func*=None)

Register, reregister, or deregister events of type 'button_press', 'button_release', or 'motion_notify'.

The specified callback function should have the following signature:

```
def func(event)
```

where event is an `MplEvent` instance containing the following data:

```
name           # Event name.
canvas         # FigureCanvas instance generating the event.
x = None      # x position - pixels from left of canvas.
y = None      # y position - pixels from bottom of canvas.
button = None  # Button pressed: None, 1, 2, 3.
key = None    # Key pressed: None, chr(range(255)), shift,
              # win, or control
inaxes = None  # Axes instance if cursor within axes.
xdata = None   # x world coordinate.
ydata = None   # y world coordinate.
```

For example:

```
def mouse_move(event):
    print event.xdata, event.ydata

a = asaplot()
a.register('motion_notify', mouse_move)
```

If `func` is `None`, the event is deregistered.

Note that in TkAgg keyboard button presses don't generate an event.

Overrides: `asap.asaplotbase.asaplotbase.register`

show(*self*, *hardrefresh*=True)

Show graphics dependent on the current buffering state.

Overrides: `asap.asaplotbase.asaplotbase.show`

terminate(*self*)

Clear the figure.

unmap(*self*)

Hide the ASAPlot graphics window.

Inherited from `asaplotbase`: `clear`, `delete`, `get_line`, `hist`, `hold`, `legend`, `palette`, `plot`, `release`, `save`, `set_axes`,

set_figure, set_limits, set_line, set_panels, set_title, subplot, text, tidy, vline_with_label

9 Module *asap.asapmath*

9.1 Functions

average_time(*args, **kwargs)

Return the (time) average of a scan or list of scans. [in channels only]

The cursor of the output scan is set to 0

Parameters:

one scan or comma separated scans or a list of scans
mask: an optional mask (only used for 'var' and 'tsys' weighting)
scanav: True averages each scan separately.
False (default) averages all scans together,
weight: Weighting scheme.
'none' (mean no weight)
'var' (1/var(spec) weighted)
'tsys' (1/Tsys**2 weighted)
'tint' (integration time weighted)
'tintsys' (Tint/Tsys**2)
'median' (median averaging)
align: align the spectra in velocity before averaging. It takes
the time of the first spectrum in the first scantable
as reference time.

Example:

```
# return a time averaged scan from scana and scanb
# without using a mask
scanav = average_time(scana,scanb)
# or equivalent
# scanav = average_time([scana, scanb])
# return the (time) averaged scan, i.e. the average of
# all correlator cycles
scanav = average_time(scan, scanav=True)
```

merge(*args)

Merge a list of scantables, or comma-separated scantables into one scantable.

Parameters:

A list [scan1, scan2] or scan1, scan2.

Example:

```
myscans = [scan1, scan2]
allscans = merge(myscans)
# or equivalent
sameallscans = merge(scan1, scan2)
```

quotient(*source*, *reference*, *preserve*=True)

Return the quotient of a 'source' (signal) scan and a 'reference' scan.
The reference can have just one scan, even if the signal has many. Otherwise they must have the same number of scans.

The cursor of the output scan is set to 0

Parameters:

source: the 'on' scan
reference: the 'off' scan
preserve: you can preserve (default) the continuum or
 remove it. The equations used are
 preserve: $\text{Output} = \text{Toff} * (\text{on/off}) - \text{Toff}$
 remove: $\text{Output} = \text{Toff} * (\text{on/off}) - \text{Ton}$

simple_math(*left*, *right*, *op*='add', *tsys*=True)

Apply simple mathematical binary operations to two scan tables, returning the result in a new scan table.

The operation is applied to both the correlations and the Tsys data

The cursor of the output scan is set to 0

Parameters:

left: the 'left' scan
right: the 'right' scan
op: the operation: 'add' (default), 'sub', 'mul', 'div'
tsys: if True (default) then apply the operation to Tsys
 as well as the data

10 Module *asap.asaplotter*

10.1 Class *asaplotter*

The ASAP plotter.

By default the plotter is set up to plot polarisations 'colour stacked' and scantables across panels.

Note:

Currently it only plots 'spectra' not Tsys or other variables.

10.1.1 Methods

```
__init__(self, visible=None)
```

```
arrow(self, *args, **kwargs)
```

Draws arrow on specified axis from (x,y) to (x+dx,y+dy).

```
axhline(self, *args, **kwargs)
```

```
AXHLINE(y=0, xmin=0, xmax=1, **kwargs)
```

Axis Horizontal Line

Draw a horizontal line at *y* from *xmin* to *xmax*. With the default values of *xmin*=0 and *xmax*=1, this line will always span the horizontal extent of the axes, regardless of the *xlim* settings, even if you change them, eg with the *xlim* command. That is, the horizontal extent is in axes coords: 0=left, 0.5=middle, 1.0=right but the *y* location is in data coordinates.

Return value is the Line2D instance. *kwargs* are the same as *kwargs* to *plot*, and can be used to control the line properties. Eg

```
# draw a thick red hline at y=0 that spans the xrange  
axhline(linewidth=4, color='r')
```

```
# draw a default hline at y=1 that spans the xrange  
axhline(y=1)
```

```
# draw a default hline at y=.5 that spans the the middle half of  
# the xrange  
axhline(y=.5, xmin=0.25, xmax=0.75)
```

```
axhspan(self, *args, **kwargs)
```

```
AXHSPAN(ymin, ymax, xmin=0, xmax=1, **kwargs)
```

Axis Horizontal Span. *y*coords are in data units and *x* coords are in axes (relative 0-1) units

Draw a horizontal span (rectangle) from *ymin* to *ymax*. With the default values of *xmin*=0 and *xmax*=1, this always span the *xrange*, regardless of the *xlim* settings, even if you change them, eg with the *xlim* command. That is, the horizontal extent is in axes coords: 0=left, 0.5=middle, 1.0=right but the *y* location is in data coordinates.

kwargs are the *kwargs* to *Patch*, eg

```
antialiased, aa  
linewidth,   lw  
edgecolor,   ec  
facecolor,   fc
```

the terms on the right are aliases

Return value is the *patches.Polygon* instance.

```
#draws a gray rectangle from y=0.25-0.75 that spans the horizontal  
#extent of the axes  
axhspan(0.25, 0.75, facecolor='0.5', alpha=0.5)
```

```
axvline(self, *args, **kwargs)
```

```
AXVLINE(x=0, ymin=0, ymax=1, **kwargs)
```

Axis Vertical Line

Draw a vertical line at x from ymin to ymax. With the default values of ymin=0 and ymax=1, this line will always span the vertical extent of the axes, regardless of the xlim settings, even if you change them, eg with the xlim command. That is, the vertical extent is in axes coords: 0=bottom, 0.5=middle, 1.0=top but the x location is in data coordinates.

Return value is the Line2D instance. kwargs are the same as kwargs to plot, and can be used to control the line properties. Eg

```
# draw a thick red vline at x=0 that spans the yrange
l = axvline(linewidth=4, color='r')

# draw a default vline at x=1 that spans the yrange
l = axvline(x=1)

# draw a default vline at x=.5 that spans the the middle half of
# the yrange
axvline(x=.5, ymin=0.25, ymax=0.75)
```

```
axvspan(self, *args, **kwargs)
```

```
AXVSPAN(xmin, xmax, ymin=0, ymax=1, **kwargs)
```

axvspan : Axis Vertical Span. xcoords are in data units and y coords are in axes (relative 0-1) units

Draw a vertical span (rectangle) from xmin to xmax. With the default values of ymin=0 and ymax=1, this always span the yrange, regardless of the ylim settings, even if you change them, eg with the ylim command. That is, the vertical extent is in axes coords: 0=bottom, 0.5=middle, 1.0=top but the y location is in data coordinates.

kwargs are the kwargs to Patch, eg

```
antialiased, aa  
linewidth,   lw  
edgecolor,   ec  
facecolor,   fc
```

the terms on the right are aliases

return value is the patches.Polygon instance.

```
# draw a vertical green translucent rectangle from x=1.25 to 1.55 that  
# spans the yrange of the axes  
axvspan(1.25, 1.55, facecolor='g', alpha=0.5)
```

```
plot(self, scan=None)
```

Plot a scantable.

Parameters:

```
scan: a scantable
```

Note:

```
If a scantable was specified in a previous call  
to plot, no argument has to be given to 'replot'  
NO checking is done that the abscissas of the scantable  
are consistent e.g. all 'channel' or all 'velocity' etc.
```

```
plot_lines(self, linecat=None, doppler=0.0, deltachan=10, rotate=90.0, location=None)
```

Plot a line catalog.

Parameters:

```

linecat:      the linecatalog to plot
doppler:      the velocity shift to apply to the frequencies
deltachan:    the number of channels to include each side of the
               line to determine a local maximum/minimum
rotate:       the rotation (in degrees) )for the text label (default 90.0)
location:     the location of the line annotation from the 'top',
               'bottom' or alternate (None - the default)

```

Notes:

If the spectrum is flagged no line will be drawn in that location.

```
save(self, filename=None, orientation=None, dpi=None)
```

Save the plot to a file. The know formats are 'png', 'ps', 'eps'.

Parameters:

```

filename:     The name of the output file. This is optional
               and autodetects the image format from the file
               suffix. If non filename is specified a file
               called 'yyyymmdd.hhmmss.png' is created in the
               current directory.
orientation:  optional parameter for postscript only (not eps).
               'landscape', 'portrait' or None (default) are valid.
               If None is choosen for 'ps' output, the plot is
               automatically oriented to fill the page.
dpi:         The dpi of the output non-ps plot

```

```
set_abcissa(self, abcissa=None, fontsize=None)
```

Set the x-axis label of the plot. If multiple panels are plotted, multiple labels have to be specified.

Parameters:

```

abcissa:      a list of abcissa labels. None (default) let
               data determine the labels

```

Example:

```

# two panels are visible on the plotter
plotter.set_ordinate(["First X-Axis", "Second X-Axis"])

```

set_colors(*self*, *colmap*)

Set the colours to be used. The plotter will cycle through these colours when lines are overlaid (stacking mode).

Parameters:

colmap: a list of colour names

Example:

```
plotter.set_colors("red green blue")
# If for example four lines are overlaid e.g I Q U V
# 'I' will be 'red', 'Q' will be 'green', U will be 'blue'
# and 'V' will be 'red' again.
```

set_colours(*self*, *colmap*)

Set the colours to be used. The plotter will cycle through these colours when lines are overlaid (stacking mode).

Parameters:

colmap: a list of colour names

Example:

```
plotter.set_colors("red green blue")
# If for example four lines are overlaid e.g I Q U V
# 'I' will be 'red', 'Q' will be 'green', U will be 'blue'
# and 'V' will be 'red' again.
```

set_font(*self*, *family*=None, *style*=None, *weight*=None, *size*=None)

Set font properties.

Parameters:

family: one of 'sans-serif', 'serif', 'cursive', 'fantasy', 'monospace'
style: one of 'normal' (or 'roman'), 'italic' or 'oblique'
weight: one of 'normal' or 'bold'
size: the 'general' font size, individual elements can be adjusted separately

set_histogram(*self*, *hist*=True, *linewidth*=None)

Enable/Disable histogram-like plotting.

Parameters:

hist: True (default) or False. The first default is taken from the .asaprc setting
plotter.histogram

```
set_layout(self, rows=None, cols=None)
```

Set the multi-panel layout, i.e. how many rows and columns plots are visible.

Parameters:

rows: The number of rows of plots
cols: The number of columns of plots

Note:

If no argument is given, the potter reverts to its auto-plot behaviour.

```
set_legend(self, mp=None, fontsize=None, mode=0)
```

Specify a mapping for the legend instead of using the default indices:

Parameters:

mp: a list of 'strings'. This should have the same length as the number of elements on the legend and then maps to the indeces in order. It is possible to uses latex math expression. These have to be enclosed in r'', e.g. r'\$x^{2}\$'

fontsize: The font size of the label (default None)

mode: where to display the legend
Any other value for loc else disables the legend:
0: auto
1: upper right
2: upper left
3: lower left
4: lower right
5: right
6: center left
7: center right
8: lower center
9: upper center
10: center

Example:

```
If the data has two IFs/rest frequencies with index 0 and 1  
for CO and Si0:  
plotter.set_stacking('i')  
plotter.set_legend(['CO', 'Si0'])  
plotter.plot()  
plotter.set_legend([r'$^{12}$CO$', r'Si0'])
```

```
set_linestyles(self, linestyles=None, linewidth=None)
```

Set the linestyles to be used. The plotter will cycle through these linestyles when lines are overlaid (stacking mode) AND only one color has been set.

Parameters:

```
linestyles:      a list of linestyles to use.
                  'line', 'dashed', 'dotted', 'dashdot',
                  'dashdotdot' and 'dashdashdot' are
                  possible
```

Example:

```
plotter.set_colors("black")
plotter.set_linestyles("line dashed dotted dashdot")
# If for example four lines are overlaid e.g I Q U V
# 'I' will be 'solid', 'Q' will be 'dashed',
# U will be 'dotted' and 'V' will be 'dashdot'.
```

```
set_mask(self, mask=None, selection=None)
```

Set a plotting mask for a specific polarization. This is useful for masking out "noise" Pangle outside a source.

Parameters:

```
mask:            a mask from scantable.create_mask
selection:       the spectra to apply the mask to.
```

Example:

```
select = selector()
select.setpolstrings("Pangle")
plotter.set_mask(mymask, select)
```

```
set_mode(self, stacking=None, panelling=None)
```

Set the plots look and feel, i.e. what you want to see on the plot.

Parameters:

```
stacking:       tell the plotter which variable to plot
                 as line colour overlays (default 'pol')
panelling:      tell the plotter which variable to plot
                 across multiple panels (default 'scan')
```

Note:

```
Valid modes are:
'beam' 'Beam' 'b':    Beams
'if' 'IF' 'i':       IFs
'pol' 'Pol' 'p':     Polarisations
'scan' 'Scan' 's':   Scans
'time' 'Time' 't':   Times
```

```
set_ordinate(self, ordinate=None, fontsize=None)
```

Set the y-axis label of the plot. If multiple panels are plotted, multiple labels have to be specified.

Parameters:

`ordinate`: a list of ordinate labels. None (default) let data determine the labels

Example:

```
# two panels are visible on the plotter
plotter.set_ordinate(["First Y-Axis", "Second Y-Axis"])
```

```
set_panelling(self, what=None)
```

```
set_range(self, xstart=None, xend=None, ystart=None, yend=None)
```

Set the range of interest on the abscissa of the plot

Parameters:

`[x,y]start, [x,y]end`: The start and end points of the 'zoom' window

Note:

These become non-sensical when the unit changes.
use `plotter.set_range()` without parameters to reset

```
set_selection(self, selection=None, refresh=True)
```

```
set_stacking(self, what=None)
```

```
set_title(self, title=None, fontsize=None)
```

Set the title of the plot. If multiple panels are plotted, multiple titles have to be specified.

Example:

```
# two panels are visible on the plotter
plotter.set_title(["First Panel", "Second Panel"])
```

```
text(self, *args, **kwargs)
```

```
TEXT(x, y, s, fontdict=None, **kwargs)
```

Add text in string *s* to axis at location *x,y* (data coords)

fontdict is a dictionary to override the default text properties.
If *fontdict* is None, the defaults are determined by your *rc*
parameters.

withdash=True will create a *TextWithDash* instance instead
of a *Text* instance.

Individual keyword arguments can be used to override any given
parameter

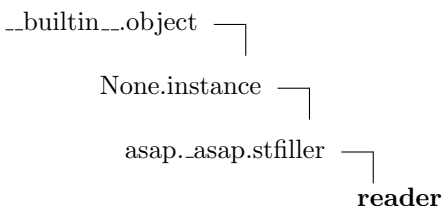
```
text(x, y, s, fontsize=12)
```

The default transform specifies that text is in data coords,
alternatively, you can specify text in axis coords (0,0 lower left and
1,1 upper right). The example below places text in the center of the
axes

```
text(0.5, 0.5, 'matplotlib',  
     horizontalalignment='center',  
     verticalalignment='center',  
     transform = ax.transAxes,  
)
```

11 Module *asap.asapreader*

11.1 Class *reader*



This class allows the user to import single dish files (rpfits,sdfits,ms).

The reader reads in integrations from the file and remains at the fileposition afterwards.

Available functions are:

```
read() # read until the (current) end of file)
```

Example:

```

r = reader('/tmp/P389.rpf')
scans r.read() # reads in the complete file into 'scans'
print scans    # summarises the contents
del r          # destroys the reader
    
```

IMPORTANT: Due to limitations in the rpfits library, only one reader can be created at a time.

```

r = reader('XYZ.rpf')
r2 = reader('ABC.rpf')
is NOT possible. This is a limitation affecting
rpfits ONLY.
    
```

11.1.1 Methods

<p>__init__(<i>self</i>, <i>filename</i>, <i>unit=None</i>, <i>theif=None</i>, <i>thebeam=None</i>)</p>
--

<p>Overrides: <i>asap._asap.stfiller.__init__</i></p>

<p>close(<i>self</i>)</p>

<p>Close the reader.</p>

<p>read(<i>self</i>)</p>

<p>Reads in all integrations in the data file.</p>
--

summary(*self*, *name=None*)

Print a summary of all scans/integrations. This reads through the whole file once.

Parameters:

 None

Example:

r.summary()

Inherited from instance: `--new--`

Inherited from object: `--delattr--`, `--getattr--`, `--hash--`, `--reduce--`, `--reduce_ex--`, `--repr--`, `--setattr--`, `--str--`

12 Module *asap.linecatalog*

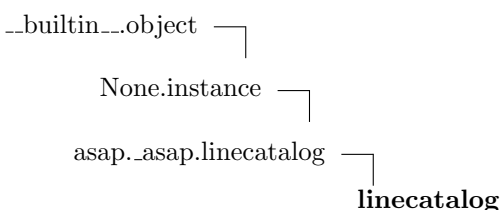
A representation of a spectra line catalog.

Author: Malte Marquarding

12.1 Variables

Name	Description
<code>__revision__</code>	Value: '\$Revision: 1259 \$' (<i>type=str</i>)

12.2 Class *linecatalog*



This class is a wrapper for line catalogs. These can be either ASCII tables or the tables saved from this class. ASCII tables have the following restrictions: Comments can be present through lines starting with '#'. The first column contains the name of the Molecule. This can't contain spaces, if it does it has to be wrapped in "" The second column contains the frequency of the transition. The third column contains the error in frequency. The fourth column contains a value describing the intensity

12.2.1 Methods

<code>__init__(self, name)</code> Overrides: <code>asap..asap.linecatalog.__init__</code>
--

<code>__getitem__(self, k)</code>

<code>__len__(self)</code>

<code>get_row(self, row=0)</code>
Get the values in a specified row of the table.
Parameters:
row: the row to retrieve

<code>reset(self)</code>
Reset the table to its initial state, i.e. undo all calls to set_
Overrides: <code>asap..asap.linecatalog.reset</code>

save(*self*, *name*, *overwrite=False*)

Save the subset of the table to disk. This uses an internal data format and can be read in again.

Overrides: `asap._asap.linecatalog.save`

set_frequency_limits(*self*, *fmin=1.0*, *fmax=120.0*, *unit='GHz'*)

Set frequency limits on the table.

Parameters:

`fmin:` the lower bound
`fmax:` the upper bound
`unit:` the frequency unit (default "GHz")

Note:

The underlying table contains frequency values in MHz

Overrides: `asap._asap.linecatalog.set_frequency_limits`

set_name(*self*, *name*, *mode='pattern'*)

Set a name restriction on the table. This can be a standard unix-style pattern or a regular expression.

Parameters:

`name:` the name pattern/regex
`mode:` the matching mode, i.e. "pattern" (default) or "regex"

Overrides: `asap._asap.linecatalog.set_name`

set_strength_limits(*self*, *smin*, *smax*)

Set line strength limits on the table (arbitrary units)

Parameters:

`smin:` the lower bound
`smax:` the upper bound

Overrides: `asap._asap.linecatalog.set_strength_limits`

summary(*self*)

Print the contents of the table.

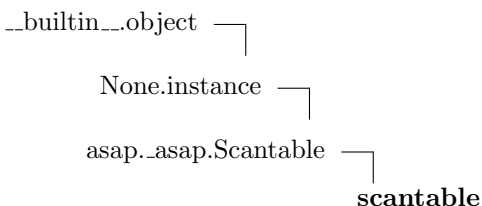
Overrides: `asap._asap.linecatalog.summary`

Inherited from instance: `__new__`

Inherited from object: `__delattr__`, `__getattr__`, `__hash__`, `__reduce__`, `__reduce_ex__`, `__repr__`, `__setattr__`, `__str__`

13 Module *asap.scantable*

13.1 Class *scantable*



The ASAP container for scans

13.1.1 Methods

<p>__init__(<i>self</i>, <i>filename</i>, <i>average</i>=None, <i>unit</i>=None)</p> <hr/> <p>Create a scantable from a saved one or make a reference</p> <p>Parameters:</p> <p>filename: the name of an asap table on disk or the name of a rpfits/sdfits/ms file (integrations within scans are auto averaged and the whole file is read) or [advanced] a reference to an existing scantable</p> <p>average: average all integrations withinb a scan on read. The default (True) is taken from .asaprc.</p> <p>unit: brightness unit; must be consistent with K or Jy. Over-rides the default selected by the reader (input rpfits/sdfits/ms) or replaces the value in existing scantables</p> <p>Overrides: <i>asap..asap.Scantable.__init__</i></p>
<p>__add__(<i>self</i>, <i>other</i>)</p>
<p>__div__(<i>self</i>, <i>other</i>)</p> <hr/> <p>implicit on all axes and on Tsys</p>
<p>__mul__(<i>self</i>, <i>other</i>)</p> <hr/> <p>implicit on all axes and on Tsys</p>

```
__str__(self)  
Overrides: __builtin__object.__str__
```

```
__sub__(self, other)  
implicit on all axes and on Tsys
```

```
add(self, offset, insitu=None)
```

Return a scan where all spectra have the offset added

Parameters:

`offset`: the offset
`insitu`: if False a new scantable is returned.
Otherwise, the scaling is done in-situ
The default is taken from `.asaprc` (False)

```
auto_poly_baseline(self, mask=[], edge=(0, 0), order=0, threshold=3, chan_avg_limit=1,  
plot=False, insitu=None)
```

Return a scan which has been baselined (all rows) by a polynomial. Spectral lines are detected first using linefinder and masked out to avoid them affecting the baseline solution.

Parameters:

mask: an optional mask retrieved from scantable

edge: an optional number of channel to drop at the edge of spectrum. If only one value is specified, the same number will be dropped from both sides of the spectrum. Default is to keep all channels. Nested tuples represent individual edge selection for different IFs (a number of spectral channels can be different)

order: the order of the polynomial (default is 0)

threshold: the threshold used by line finder. It is better to keep it large as only strong lines affect the baseline solution.

chan_avg_limit: a maximum number of consecutive spectral channels to average during the search of weak and broad lines. The default is no averaging (and no search for weak lines). If such lines can affect the fitted baseline (e.g. a high order polynomial is fitted), increase this parameter (usually values up to 8 are reasonable). Most users of this method should find the default value sufficient.

plot: plot the fit and the residual. In this each individual fit has to be approved, by typing 'y' or 'n'

insitu: if False a new scantable is returned. Otherwise, the scaling is done in-situ. The default is taken from .asaprc (False)

Example:

```
scan2=scan.auto_poly_baseline(order=7)
```

auto_quotient(*self*, *preserve*=True, *mode*='paired')

This function allows to build quotients automatically.
It assumes the observation to have the same number of
"ons" and "offs"

Parameters:

preserve: you can preserve (default) the continuum or
remove it. The equations used are
preserve: $\text{Output} = \text{Toff} * (\text{on/off}) - \text{Toff}$
remove: $\text{Output} = \text{Toff} * (\text{on/off}) - \text{Ton}$

mode: the on/off detection mode
'paired' (default)
identifies 'off' scans by the
trailing 'R' (Mopra/Parkes) or
'_e'/'_w' (Tid) and matches
on/off pairs from the observing pattern

'time'
finds the closest off in time

average_beam(*self*, *mask*=None, *weight*='none')

Average the Beams together.

Parameters:

mask: An optional mask defining the region, where the
averaging will be applied. The output will have all
specified points masked.

weight: Weighting scheme. 'none' (default), 'var' (1/var(spec)
weighted), or 'tsys' (1/Tsys**2 weighted)

average_pol(*self*, *mask*=None, *weight*='none')

Average the Polarisation together.

Parameters:

mask: An optional mask defining the region, where the
averaging will be applied. The output will have all
specified points masked.

weight: Weighting scheme. 'none' (default), 'var' (1/var(spec)
weighted), or 'tsys' (1/Tsys**2 weighted)

average_time(*self*, *mask*=None, *scanav*=False, *weight*='tint', *align*=False)

Return the (time) weighted average of a scan.

Note:

in channels only - align if necessary

Parameters:

mask: an optional mask (only used for 'var' and 'tsys' weighting)

scanav: True averages each scan separately
False (default) averages all scans together,

weight: Weighting scheme.
'none' (mean no weight)
'var' (1/var(spec) weighted)
'tsys' (1/Tsys**2 weighted)
'tint' (integration time weighted)
'tintsys' (Tint/Tsys**2)
'median' (median averaging)
The default is 'tint'

align: align the spectra in velocity before averaging. It takes the time of the first spectrum as reference time.

Example:

```
# time average the scantable without using a mask
newscan = scan.average_time()
```

bin(*self*, *width*=5, *insitu*=None)

Return a scan where all spectra have been binned up.

Parameters:

width: The bin width (default=5) in pixels
insitu: if False a new scantable is returned.
Otherwise, the scaling is done in-situ
The default is taken from .asaprc (False)

convert_flux(*self*, *jyperk*=None, *eta*=None, *d*=None, *insitu*=None)

Return a scan where all spectra are converted to either

Jansky or Kelvin depending upon the flux units of the scan table.

By default the function tries to look the values up internally.

If it can't find them (or if you want to over-ride), you must specify EITHER jyperk OR eta (and D which it will try to look up also if you don't set it). jyperk takes precedence if you set both.

Parameters:

jyperk: the Jy / K conversion factor
eta: the aperture efficiency
d: the geomtric diameter (metres)
insitu: if False a new scantable is returned.
Otherwise, the scaling is done in-situ
The default is taken from .asaprc (False)

convert_pol(*self*, *poltype*=None)

Convert the data to a different polarisation type.

Parameters:

poltype: The new polarisation type. Valid types are:
"linear", "stokes" and "circular"

copy(*self*)

Return a copy of this scantable.

Note:

This makes a full (deep) copy. `scan2 = scan1` makes a reference.

Parameters:

none

Example:

```
copiedscan = scan.copy()
```

create_mask(*self*, **args*, ***kwargs*)

Compute and return a mask based on [min, max] windows.

The specified windows are to be INCLUDED, when the mask is applied.

Parameters:

[min, max], [min2, max2], ...

Pairs of start/end points (inclusive) specifying the regions to be masked

invert: optional argument. If specified as True, return an inverted mask, i.e. the regions specified are EXCLUDED

row: create the mask using the specified row for unit conversions, default is `row=0` only necessary if frequency varies over rows.

Example:

```
scan.set_unit('channel')
```

a)

```
msh = scan.create_mask([400, 500], [800, 900])
```

```
# masks everything outside 400 and 500
```

```
# and 800 and 900 in the unit 'channel'
```

b)

```
msh = scan.create_mask([400, 500], [800, 900], invert=True)
```

```
# masks the regions between 400 and 500
```

```
# and 800 and 900 in the unit 'channel'
```

c)

```
mask only channel 400
```

```
msh = scan.create_mask([400, 400])
```

drop_scan(*self*, *scanid*=None)

Return a new scantable where the specified scan number(s) has(have) been dropped.

Parameters:

scanid: a (list of) scan number(s)

flag(*self*, *mask*=None)

Flag the selected data using an optional channel mask.

Parameters:

mask: an optional channel mask, created with `create_mask`. Default (no mask) is all channels.

freq_align(*self*, *reftime*=None, *method*='cubic', *insitu*=None)

Return a scan where all rows have been aligned in frequency/velocity. The alignment frequency frame (e.g. LSRK) is that set by function `set_freqframe`.

Parameters:

reftime: reference time to align at. By default, the time of the first row of data is used.

method: Interpolation method for regridding the spectra. Choose from "nearest", "linear", "cubic" (default) and "spline"

insitu: if False a new scantable is returned. Otherwise, the scaling is done in-situ. The default is taken from `.asaprc` (False)

freq_switch(*self*, *insitu*=None)

Apply frequency switching to the data.

Parameters:

insitu: if False a new scantable is returned. Otherwise, the switching is done in-situ. The default is taken from `.asaprc` (False)

Example:

none

```
gain_el(self, poly=None, filename='', method='linear', insitu=None)
```

Return a scan after applying a gain-elevation correction.
 The correction can be made via either a polynomial or a table-based interpolation (and extrapolation if necessary).
 You specify polynomial coefficients, an ascii table or neither.
 If you specify neither, then a polynomial correction will be made with built in coefficients known for certain telescopes (an error will occur if the instrument is not known).
 The data and Tsys are **divided** by the scaling factors.

Parameters:

poly: Polynomial coefficients (default None) to compute a gain-elevation correction as a function of elevation (in degrees).

filename: The name of an ascii file holding correction factors. The first row of the ascii file must give the column names and these MUST include columns "ELEVATION" (degrees) and "FACTOR" (multiply data by this) somewhere. The second row must give the data type of the column. Use 'R' for Real and 'I' for Integer. An example file would be (actual factors are arbitrary) :

```
TIME ELEVATION FACTOR
```

```
R R R
```

```
0.1 0 0.8
```

```
0.2 20 0.85
```

```
0.3 40 0.9
```

```
0.4 60 0.85
```

```
0.5 80 0.8
```

```
0.6 90 0.75
```

method: Interpolation method when correcting from a table. Values are "nearest", "linear" (default), "cubic" and "spline"

insitu: if False a new scantable is returned. Otherwise, the scaling is done in-situ. The default is taken from .asaprc (False)

```
get_abcissa(self, rowno=0)
```

Get the abcissa in the current coordinate setup for the currently selected Beam/IF/Pol

Parameters:

rowno: an optional row number in the scantable. Default is the first row, i.e. rowno=0

Returns:

The abcissa values and the format string (as a dictionary)

get_azimuth(*self*, *row*=-1)

Get a list of azimuths for the observations.
Return a float for each integration in the scantable.

Parameters:

row: row no of integration. Default -1 return all rows.

Example:

none

get_column_names(*self*)

Return a list of column names, which can be used for selection.

Overrides: *asap._asap.Scantable.get_column_names*

get_direction(*self*, *row*=-1)

Get a list of Positions on the sky (direction) for the observations.
Return a float for each integration in the scantable.

Parameters:

row: row no of integration. Default -1 return all rows

Example:

none

get_elevation(*self*, *row*=-1)

Get a list of elevations for the observations.
Return a float for each integration in the scantable.

Parameters:

row: row no of integration. Default -1 return all rows.

Example:

none

get_fit(*self*, *row*=0)

Print or return the stored fits for a row in the scantable

Parameters:

row: the row which the fit has been applied to.

get_inttime(*self*, *row*=-1)

Get a list of integration times for the observations.
Return a time in seconds for each integration in the scantable.

Parameters:

row: row no of integration. Default -1 return all rows.

Example:

none

get_parangle(*self*, *row*=-1)

Get a list of parallactic angles for the observations.

Return a float for each integration in the scantable.

Parameters:

row: row no of integration. Default -1 return all rows.

Example:

 none

get_restfreqs(*self*)

Get the restfrequency(s) stored in this scantable.

The return value(s) are always of unit 'Hz'

Parameters:

 none

Returns:

 a list of doubles

get_scan(*self*, *scanid*=None)

Return a specific scan (by *scanno*) or collection of scans (by source name) in a new scantable.

Note:

 See `scantable.drop_scan()` for the inverse operation.

Parameters:

scanid: a (list of) *scanno* or a source name, unix-style
 patterns are accepted for source name matching, e.g.
 '*R' gets all 'ref scans'

Example:

```
# get all scans containing the source '323p459'
newscan = scan.get_scan('323p459')
# get all 'off' scans
refscans = scan.get_scan('*R')
# get a subset of scans by scanno (as listed in scan.summary())
newscan = scan.get_scan([0, 2, 7, 10])
```

get_selection(*self*)

Get the selection object currently set on this scantable.

Parameters:

 none

Example:

```
sel = scan.get_selection()
sel.set_ifs(0)          # select IF 0
scan.set_selection(sel)  # apply modified selection
```

get_sourcename(*self*, *row*=-1)

Get a list source names for the observations.
Return a string for each integration in the scantable.

Parameters:

row: row no of integration. Default -1 return all rows.

Example:

 none

get_time(*self*, *row*=-1, *asdatetime*=False)

Get a list of time stamps for the observations.
Return a datetime object for each integration time stamp in the scantable.

Parameters:

row: row no of integration. Default -1 return all rows

asdatetime: return values as datetime objects rather than strings

Example:

 none

get_tsys(*self*)

Return the System temperatures.

Returns:

 a list of Tsys values for the current selection

get_unit(*self*)

Get the default unit set in this scantable

Returns:

 A unit string

history(*self*, *filename*=None)

Print the history. Optionally to a file.

Parameters:

filename: The name of the file to save the history to.

invert_phase(*self*)

Invert the phase of the complex polarisation

lag_flag(*self*, *frequency*, *width*=0.0, *unit*='GHz', *insitu*=None)

Flag the data in 'lag' space by providing a frequency to remove.

Flagged data in the scantable gets set to 0.0 before the fft.

No taper is applied.

Parameters:

frequency: the frequency (really a period within the bandwidth) to remove

width: the width of the frequency to remove, to remove a range of frequencies around the centre.

unit: the frequency unit (default "GHz")

Notes:

It is recommended to flag edges of the band or strong signals beforehand.

mx_quotient(*self*, *mask*=None, *weight*='median', *preserve*=True)

Form a quotient using "off" beams when observing in "MX" mode.

Parameters:

mask: an optional mask to be used when *weight* == 'stddev'

weight: How to average the off beams. Default is 'median'.

preserve: you can preserve (default) the continuum or remove it. The equations used are

preserve: Output = Toff * (on/off) - Toff

remove: Output = Toff * (on/off) - Ton

opacity(*self*, *tau*, *insitu*=None)

Apply an opacity correction. The data and Tsys are multiplied by the correction factor.

Parameters:

tau: Opacity from which the correction factor is $\exp(\tau * ZD)$ where ZD is the zenith-distance

insitu: if False a new scantable is returned. Otherwise, the scaling is done in-situ. The default is taken from .asaprc (False)

poly_baseline(*self*, *mask*=None, *order*=0, *plot*=False, *insitu*=None)

Return a scan which has been baselined (all rows) by a polynomial.

Parameters:

mask: an optional mask
order: the order of the polynomial (default is 0)
plot: plot the fit and the residual. In this each individual fit has to be approved, by typing 'y' or 'n'
insitu: if False a new scantable is returned. Otherwise, the scaling is done in-situ. The default is taken from .asaprc (False)

Example:

```
# return a scan baselined by a third order polynomial,
# not using a mask
bscan = scan.poly_baseline(order=3)
```

recalc_azel(*self*)

Recalculate the azimuth and elevation for each position.

Parameters:

none

Example:

resample(*self*, *width*=5, *method*='cubic', *insitu*=None)

Return a scan where all spectra have been binned up.

Parameters:

width: The bin width (default=5) in pixels
method: Interpolation method when correcting from a table. Values are "nearest", "linear", "cubic" (default) and "spline"
insitu: if False a new scantable is returned. Otherwise, the scaling is done in-situ. The default is taken from .asaprc (False)

rotate_linpolphase(*self*, *angle*)

Rotate the phase of the complex polarization $O=Q+iU$ correlation.

This is always done in situ in the raw data. So if you call this function more than once then each call rotates the phase further.

Parameters:

angle: The angle (degrees) to rotate (add) by.

Examples:

```
scan.rotate_linpolphase(2.3)
```

rotate_xyphase(*self*, *angle*)

Rotate the phase of the XY correlation. This is always done in situ in the data. So if you call this function more than once then each call rotates the phase further.

Parameters:

angle: The angle (degrees) to rotate (add) by.

Examples:

```
scan.rotate_xyphase(2.3)
```

save(*self*, *name*=None, *format*=None, *overwrite*=False)

Store the scantable on disk. This can be an asap (aips++) Table, SDFITS or MS2 format.

Parameters:

name: the name of the outputfile. For format "ASCII" this is the root file name (data in 'name'.txt and header in 'name'_header.txt)

format: an optional file format. Default is ASAP. Allowed are - 'ASAP' (save as ASAP [aips++] Table), 'SDFITS' (save as SDFITS file) 'ASCII' (saves as ascii text file) 'MS2' (saves as an aips++ MeasurementSet V2)

overwrite: If the file should be overwritten if it exists. The default False is to return with warning without writing the output. USE WITH CARE.

Example:

```
scan.save('myscan.asap')
scan.save('myscan.sdfits', 'SDFITS')
```

scale(*self*, *factor*, *tsys*=True, *insitu*=None)

Return a scan where all spectra are scaled by the give 'factor'

Parameters:

factor: the scaling factor

insitu: if False a new scantable is returned. Otherwise, the scaling is done in-situ. The default is taken from .asaprc (False)

tsys: if True (default) then apply the operation to Tsys as well as the data

set_dirframe(*self*, *frame*='')

Set the frame type of the Direction on the sky.

Parameters:

frame: an optional frame type, default ''. Valid frames are:
'J2000', 'B1950', 'GALACTIC'

Examples:

```
scan.set_dirframe('GALACTIC')
```

Overrides: *asap.asap.Scantable.set_dirframe*

set_doppler(*self*, *doppler*='RADIO')

Set the doppler for all following operations on this scantable.

Parameters:

doppler: One of 'RADIO', 'OPTICAL', 'Z', 'BETA', 'GAMMA'

set_feedtype(*self*, *feedtype*)

Overwrite the feed type, which might not be set correctly.

Parameters:

feedtype: 'linear' or 'circular'

set_freqframe(*self*, *frame*=None)

Set the frame type of the Spectral Axis.

Parameters:

frame: an optional frame type, default 'LSRK'. Valid frames are:
'REST', 'TOPO', 'LSRD', 'LSRK', 'BARY',
'GEO', 'GALACTO', 'LGROUP', 'CMB'

Examples:

```
scan.set_freqframe('BARY')
```

set_instrument(*self*, *instr*)

Set the instrument for subsequent processing.

Parameters:

instr: Select from 'ATPKSMB', 'ATPKSHOH', 'ATMOPRA',
'DSS-43' (Tid), 'CEDUNA', and 'HOBART'

```
set_restfreqs(self, freqs=None, unit='Hz')
```

Set or replace the restfrequency specified and
If the 'freqs' argument holds a scalar,
then that rest frequency will be applied to all the selected
data. If the 'freqs' argument holds
a vector, then it MUST be of equal or smaller length than
the number of IFs (and the available restfrequencies will be
replaced by this vector). In this case, **all** data have
the restfrequency set per IF according
to the corresponding value you give in the 'freqs' vector.
E.g. 'freqs=[1e9, 2e9]' would mean IF 0 gets restfreq 1e9 and
IF 1 gets restfreq 2e9.
You can also specify the frequencies via a linecatalog/

Parameters:

```
freqs: list of rest frequency values or string identifiers  
unit: unit for rest frequency (default 'Hz')
```

Example:

```
# set the given restfrequency for the whole table  
scan.set_restfreqs(freqs=1.4e9)  
# If the number of IFs in the data is >= 2 IF0 gets the first  
# value IF1 the second...  
scan.set_restfreqs(freqs=[1.4e9, 1.67e9])  
#set the given restfrequency for the whole table (by name)  
scan.set_restfreqs(freqs="OH1667")
```

Note:

```
To do more sophisticated Restfrequency setting, e.g. on a  
source and IF basis, use scantable.set_selection() before using  
this function.  
# provide your scantable is call scan  
selection = selector()  
selection.set_name("ORION*")  
selection.set_ifs([1])  
scan.set_selection(selection)  
scan.set_restfreqs(freqs=86.6e9)
```

```
set_selection(self, selection=<asap.selector.selector object at 0x40221eb4>)
```

Select a subset of the data. All following operations on this scantable are only applied to thi selection.

Parameters:

 selection: a selector object (default unset the selection)

Examples:

```
sel = selector()           # create a selection object
self.set_scans([0, 3])    # select SCANNO 0 and 3
scan.set_selection(sel)   # set the selection
scan.summary()           # will only print summary of scanno 0 an 3
scan.set_selection()     # unset the selection
```

```
set_unit(self, unit='channel')
```

Set the unit for all following operations on this scantable

Parameters:

 unit: optional unit, default is 'channel'
 one of '*Hz', 'km/s', 'channel', ''

```
shift_refpix(self, delta)
```

Shift the reference pixel of the Spectra Coordinate by an integer amount.

Parameters:

 delta: the amount to shift by

Note:

 Be careful using this with broadband data.

Overrides: `asap._asap.Scantable.shift_refpix`

```
smooth(self, kernel='hanning', width=5.0, insitu=None)
```

Smooth the spectrum by the specified kernel (conserving flux).

Parameters:

 kernel: The type of smoothing kernel. Select from 'hanning' (default), 'gaussian' and 'boxcar'. The first three characters are sufficient.

 width: The width of the kernel in pixels. For hanning this is ignored otherwise it defaults to 5 pixels. For 'gaussian' it is the Full Width Half Maximum. For 'boxcar' it is the full width.

 insitu: if False a new scantable is returned. Otherwise, the scaling is done in-situ. The default is taken from `.asaprc` (False)

Example:

 none

stats(*self*, *stat*='stddev', *mask*=None)

Determine the specified statistic of the current beam/if/pol
 Takes a 'mask' as an optional parameter to specify which
 channels should be excluded.

Parameters:

```

stat:    'min', 'max', 'sumsq', 'sum', 'mean'
         'var', 'stddev', 'avdev', 'rms', 'median'
mask:    an optional mask specifying where the statistic
         should be determined.
  
```

Example:

```

scan.set_unit('channel')
msk = scan.create_mask([100, 200], [500, 600])
scan.stats(stat='mean', mask=m)
  
```

stddev(*self*, *mask*=None)

Determine the standard deviation of the current beam/if/pol
 Takes a 'mask' as an optional parameter to specify which
 channels should be excluded.

Parameters:

```

mask:    an optional mask specifying where the standard
         deviation should be determined.
  
```

Example:

```

scan.set_unit('channel')
msk = scan.create_mask([100, 200], [500, 600])
scan.stddev(mask=m)
  
```

summary(*self*, *filename*=None)

Print a summary of the contents of this scantable.

Parameters:

```

filename: the name of a file to write the putput to
          Default - no file output
verbose:  print extra info such as the frequency table
          The default (False) is taken from .asaprc
  
```

swap_linears(*self*)

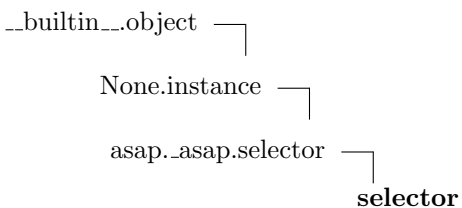
Swap the linear polarisations XX and YY, or better the first two polarisations as this also works for
 circulars.

Inherited from instance: `__new__`

Inherited from object: `__delattr__`, `__getattr__`, `__hash__`, `__reduce__`, `__reduce_ex__`, `__repr__`, `__setattr__`

14 Module *asap.selector*

14.1 Class selector



A selection object to be applied to scantables to restrict the scantables to specific rows.

14.1.1 Methods

<code>__add__(self, other)</code> Merge two selections.
<code>__str__(self)</code> Overrides: <code>__builtin__.object.__str__</code>
<code>get_beams(self)</code>
<code>get_cycles(self)</code>
<code>get_ifs(self)</code>
<code>get_name(self)</code>
<code>get_order(self)</code>
<code>get_pols(self)</code>
<code>get_poltypes(self)</code>
<code>get_query(self)</code>
<code>get_scans(self)</code>
<code>is_empty(self)</code> Has anything been set?

reset(*self*)

Unset all selections.

set_beams(*self*, *beams*=[])

Set a sequence of Beam numbers (0-based).

Parameters:

`beams:` a list of integers. Default [] is to unset the selection.

set_cycles(*self*, *cycles*=[])

Set a sequence of IF numbers (0-based).

Parameters:

`cycles:` a list of integers. Default [] is to unset the selection.

set_ifs(*self*, *ifs*=[])

Set a sequence of IF numbers (0-based).

Parameters:

`ifs:` a list of integers. Default [] is to unset the selection.

set_name(*self*, *name*)

Set a selection based on a name. This can be a unix pattern , e.g. "*R"

Parameters:

`name:` a string containing a source name or pattern

Examples:

```
# select all reference scans which start with "Orion"
selection.set_name("Orion*R")
```

set_order(*self*, *order*)

Set the order the scantable should be sorted by.

Parameters:

`order:` The list of column names to sort by in order

set_polarisations(*self*, *pols*=[])

Set the polarisations to be selected in the scantable.

Parameters:

pols: a list of integers of 0-3, or strings, e.g ["I","Q"].
Default [] is no selection

Example:

```
sel = selector()
# These are equivalent if data is 'linear'
sel.set_polarisations(["XX","Re(XY)"])
sel.set_polarisations([0,2])
# reset the polarisation selection
sel.set_polarisations()
```

set_polarization(*self*, *pols*=[])

Set the polarisations to be selected in the scantable.

Parameters:

pols: a list of integers of 0-3, or strings, e.g ["I","Q"].
Default [] is no selection

Example:

```
sel = selector()
# These are equivalent if data is 'linear'
sel.set_polarisations(["XX","Re(XY)"])
sel.set_polarisations([0,2])
# reset the polarisation selection
sel.set_polarisations()
```

set_query(*self*, *query*)

Select by Column query. Power users only!

Example:

```
# select all off scans with integration times over 60 seconds.
selection.set_query("SRCTYPE == 1 AND INTERVAL > 60.0")
```

set_scans(*self*, *scans*=[])

Set a sequence of Scan numbers (0-based).

Parameters:

scans: a list of integers. Default [] is to unset the selection.

```
set_tsys(self, tsysmin=0.0, tsysmax=None)
```

Select by Tsys range.

Parameters:

 tsysmin: the lower threshold. Default 0.0
 tsysmax: the upper threshold. Default None.

Examples:

```
# select all spectra with Tsys <= 500.0  
selection.set_tsys(tsysmax=500.0)
```

Inherited from instance: `--new--`

Inherited from object: `--delattr--`, `--getattr--`, `--hash--`, `--reduce--`, `--reduce_ex--`, `--repr--`, `--setattr--`

Index

- asap (*package*), 2–4
 - is_ipython (*function*), 2
 - list_files (*function*), 2
 - list_rcparameters (*function*), 2
 - mask_and (*function*), 2
 - mask_not (*function*), 2
 - mask_or (*function*), 3
 - print_log (*function*), 3
 - rc (*function*), 3
 - rc_params (*function*), 3
 - rcdefaults (*function*), 3
 - unique (*function*), 3
 - welcome (*function*), 3
- asap.asapfit (*module*), 5
 - asapfit (*class*), 5
 - __init__ (*method*), 5
 - __str__ (*method*), 5
 - as_dict (*method*), 5
 - save (*method*), 5
- asap.asapfitter (*module*), 6–9
 - fitter (*class*), 6–9
 - __init__ (*method*), 6
 - auto_fit (*method*), 6
 - commit (*method*), 6
 - fit (*method*), 6
 - get_area (*method*), 6
 - get_chi2 (*method*), 6
 - get_errors (*method*), 7
 - get_estimate (*method*), 7
 - get_fit (*method*), 7
 - get_parameters (*method*), 7
 - get_residual (*method*), 7
 - plot (*method*), 7
 - set_data (*method*), 7
 - set_function (*method*), 8
 - set_gauss_parameters (*method*), 8
 - set_parameters (*method*), 8
 - set_scan (*method*), 8
 - store_fit (*method*), 9
- asap.asaplinefind (*module*), 10–12
 - linefinder (*class*), 10–12
 - __init__ (*method*), 10
 - find_lines (*method*), 10
 - get_mask (*method*), 11
 - get_ranges (*method*), 11
 - set_options (*method*), 11
 - set_scan (*method*), 12
- asap.asaplot (*module*), 13
 - asaplot (*class*), 13
 - __init__ (*method*), 13
- asap.asaplotbase (*module*), 14–18
 - asaplotbase (*class*), 14–18
 - __init__ (*method*), 14
 - clear (*method*), 14
 - delete (*method*), 14
 - get_line (*method*), 14
 - hist (*method*), 14
 - hold (*method*), 14
 - legend (*method*), 14
 - palette (*method*), 15
 - plot (*method*), 15
 - position (*method*), 15
 - region (*method*), 15
 - register (*method*), 15
 - release (*method*), 16
 - save (*method*), 16
 - set_axes (*method*), 16
 - set_figure (*method*), 17
 - set_limits (*method*), 17
 - set_line (*method*), 17
 - set_panels (*method*), 17
 - set_title (*method*), 17
 - show (*method*), 18
 - subplot (*method*), 18
 - text (*method*), 18
 - tidy (*method*), 18
 - vline_with_label (*method*), 18
- asap.asaplotgui (*module*), 19–21
 - asaplotgui (*class*), 19–21
 - __init__ (*method*), 19
 - map (*method*), 19
 - position (*method*), 19
 - quit (*method*), 19
 - region (*method*), 19
 - register (*method*), 19
 - show (*method*), 20
 - terminate (*method*), 20
 - unmap (*method*), 20
- asap.asaplotgui_gtk (*module*), 22–24
 - asaplotgui (*class*), 22–24
 - __init__ (*method*), 22
 - map (*method*), 22
 - position (*method*), 22
 - quit (*method*), 22

- region (*method*), 22
- register (*method*), 22
- show (*method*), 23
- terminate (*method*), 23
- unmap (*method*), 23
- asap.asapmath (*module*), 25–26
 - average_time (*function*), 25
 - merge (*function*), 25
 - quotient (*function*), 25
 - simple_math (*function*), 26
- asap.asapplotter (*module*), 27–36
 - asapplotter (*class*), 27–36
 - __init__ (*method*), 27
 - arrow (*method*), 27
 - axhline (*method*), 27
 - axhspan (*method*), 27
 - axvline (*method*), 28
 - axvspan (*method*), 29
 - plot (*method*), 30
 - plot_lines (*method*), 30
 - save (*method*), 31
 - set_abcissa (*method*), 31
 - set_colors (*method*), 31, 32
 - set_font (*method*), 32
 - set_histogram (*method*), 32
 - set_layout (*method*), 32
 - set_legend (*method*), 33
 - set_linestyles (*method*), 33
 - set_mask (*method*), 34
 - set_mode (*method*), 34
 - set_ordinate (*method*), 34
 - set_panelling (*method*), 35
 - set_range (*method*), 35
 - set_selection (*method*), 35
 - set_stacking (*method*), 35
 - set_title (*method*), 35
 - text (*method*), 35
- asap.asapreader (*module*), 37–38
 - reader (*class*), 37–38
 - __init__ (*method*), 37
 - close (*method*), 37
 - read (*method*), 37
 - summary (*method*), 37
- asap.linecatalog (*module*), 39–40
 - linecatalog (*class*), 39–40
 - __getitem__ (*method*), 39
 - __init__ (*method*), 39
 - __len__ (*method*), 39
 - get_row (*method*), 39
 - reset (*method*), 39
 - save (*method*), 39
 - set_frequency_limits (*method*), 40
 - set_name (*method*), 40
 - set_strength_limits (*method*), 40
 - summary (*method*), 40
- asap.scantable (*module*), 41–58
 - scantable (*class*), 41–58
 - __add__ (*method*), 41
 - __div__ (*method*), 41
 - __init__ (*method*), 41
 - __mul__ (*method*), 41
 - __str__ (*method*), 41
 - __sub__ (*method*), 42
 - add (*method*), 42
 - auto_poly_baseline (*method*), 42
 - auto_quotient (*method*), 43
 - average_beam (*method*), 44
 - average_pol (*method*), 44
 - average_time (*method*), 44
 - bin (*method*), 45
 - convert_flux (*method*), 45
 - convert_pol (*method*), 45
 - copy (*method*), 46
 - create_mask (*method*), 46
 - drop_scan (*method*), 46
 - flag (*method*), 47
 - freq_align (*method*), 47
 - freq_switch (*method*), 47
 - gain_el (*method*), 47
 - get_abcissa (*method*), 48
 - get_azimuth (*method*), 48
 - get_column_names (*method*), 49
 - get_direction (*method*), 49
 - get_elevation (*method*), 49
 - get_fit (*method*), 49
 - get_inttime (*method*), 49
 - get_parangle (*method*), 49
 - get_restfreqs (*method*), 50
 - get_scan (*method*), 50
 - get_selection (*method*), 50
 - get_sourcename (*method*), 50
 - get_time (*method*), 51
 - get_tsys (*method*), 51
 - get_unit (*method*), 51
 - history (*method*), 51
 - invert_phase (*method*), 51

lag_flag (*method*), 51
mx_quotient (*method*), 52
opacity (*method*), 52
poly_baseline (*method*), 52
recalc_azel (*method*), 53
resample (*method*), 53
rotate_linpolphase (*method*), 53
rotate_xyphase (*method*), 53
save (*method*), 54
scale (*method*), 54
set_dirframe (*method*), 54
set_doppler (*method*), 55
set_feedtype (*method*), 55
set_freqframe (*method*), 55
set_instrument (*method*), 55
set_restfreqs (*method*), 55
set_selection (*method*), 56
set_unit (*method*), 57
shift_refpix (*method*), 57
smooth (*method*), 57
stats (*method*), 57
stddev (*method*), 58
summary (*method*), 58
swap_linears (*method*), 58
asap.selector (*module*), 59–62
selector (*class*), 59–62
 __add__ (*method*), 59
 __str__ (*method*), 59
 get_beams (*method*), 59
 get_cycles (*method*), 59
 get_ifs (*method*), 59
 get_name (*method*), 59
 get_order (*method*), 59
 get_pols (*method*), 59
 get_poltypes (*method*), 59
 get_query (*method*), 59
 get_scans (*method*), 59
 is_empty (*method*), 59
 reset (*method*), 59
 set_beams (*method*), 60
 set_cycles (*method*), 60
 set_ifs (*method*), 60
 set_name (*method*), 60
 set_order (*method*), 60
 set_polarisations (*method*), 60, 61
 set_query (*method*), 61
 set_scans (*method*), 61
 set_tsys (*method*), 61