CASA Toolkit Reference Manual

CASA Group, eds

March 30, 2016

CASA Reference Manual.

The CASA Reference Manual contains the documentation on the tool-based functionality within the system. There are five broad packages:

- General- modules that are of general use for astronomical processing
  - Images - create, manipulate and analyze images
  - Coordsys - functionality for manipulating coordinate systems
  - Componentlist - functionality for manipulating components
  - MeasurementSet - functionality for manipulating MeasurementSets (CASA data structures)
  - Measures - functionality for handling quantities with a specified reference frame
  - Quanta - functionality for handling quantities (values with units)
- Synthesis - modules needed for processing synthesis data
  - Calibrater - synthesis calibration facilities
  - AgentFlagger - flagging framework for manual or automatic flagging
  - Imager - synthesis and single dish imaging including deconvolution
  - Simulator - facilities for simulation of telescope data
  - vpmanager - facilities for specifying voltage patterns and primary beams
- Utility - non-astronomy specific functionality
  - Table - functionality for manipulating tables in CASA
- Third Party - modules that interface to 3rd party packages
  - Atmosphere - Interface to Juan R. Padro's Atmospheric Transmission Model (ATM) library.
- Single-dish - functionality for processing single-dish data
  - ASAP - single-dish data analysis package

Contents

Chapter 1

Package General

The general package contains modules that are of general use for astronomical processing.

images-Module.html

1.1 images - Module

Access and analysis of images

Description

This module contains functionality to access, create, and analyze CASA images. It offers both basic services and higher-level packaged tools.
The available tools in this module are

- Image - create, manipulate and analyze images (default tool is **ia**). This tool provides a range of low to medium level services.
- Regionmanager - create and manipulate `regions-of-interest` (default tool is **rg**). **WARNING! Documentation describes Glish-based tool which has been only partially ported to CASA.**
- Coordsys - create and manipulate Coordinate Systems (default tool is **cs**).
- Imagepol - this offers specialized polarimetric analysis of images.

Images

We refer to a CASA `image file` when we are referring to the actual data stored on disk. The name that you give a CASA `image file` is actually the name of a directory containing a collection of CASA tables which together constitute the `image file`. But you only need to refer to the directory name and you can think of it as one *logical* file. Some images don't have an associated disk file. These are called "virtual" images (e.g. an image that is temporary and

entirely in memory). Whenever we use the word "image", we are just using it in a generic sense.

Images are manipulated with an Image tool.

Pixel mask

A `pixel mask` specifies which pixels are to be considered good (mask value `T`) or bad (mask value `F`). For example, you may have imported a FITS file which has blanked pixels in it. These will be converted into `pixel mask` elements whose values are bad (`F`). Or you may have made an error analysis of an image and computed via a statistical test that certain pixels should be masked out for future analysis.

An `image file` may contain zero, one, or more `pixel masks`. However, only one mask will be designated as the default mask and be applied to the data.

For more details, see the Image tool.

Region-of-interest

A `region-of-interest` or simply, region, designates which pixels of the image you are interested in for some (generally) astrophysical reason. This complements the `pixel mask` which specifies which pixels are good or bad (for statistical reasons). `Regions-of-interest` are generated and manipulated with the Regionmanager tool.

Briefly, a `region-of-interest` may be either a simple shape such as a multi-dimensional box, or a 2-D polygon, or some compound combination of `regions-of-interest`. For example, a 2-D polygon defined in the X and Y axes extended along the Z axis, or perhaps a union or intersection of regions.

See the Regionmanager documentation for more details on regions.

Coordinates

We will often refer to (absolute) pixel coordinates. Consider a 2-D image with shape [10,20]. Then our model is that the centre of the bottom-left corner pixel has pixel coordinate [X,Y] = [0,0]. The centre of the top-right corner pixel has pixel coordinate [X,Y] = [9,19].

When a physical Coordinate System (e.g. an RA/DEC direction coordinate) is attached to an image, then we can convert pixel coordinates to a world (or physical) coordinate.

The Coordsys `tool` is available for manipulating Coordinate Systems. You can recover the Coordinate System from an image into a Coordsys `tool` via the Image function coordsys.

For more details, see the Image and Coordsys tools.

Lattice Expression Language (LEL)

This allows you to manipulate expressions involving images. For example, add this image to that image, or multiply the miniumum value of that image by the square root of this image. The LEL syntax is quite rich and is described in detail in note 223.

LEL is used in several of the Image tool functions.

Example Here is a simple example to give you the flavour of using the `image` module. Suppose we have an image FITS disk file; we would like to convert it to a CASA `image file` (store the image in a CASA table), look at the header information and store it in a record for future use, work out some statistics and then close the image.

```
"""
#
print "\t----\t Module Ex 1 \t----"
pathname=os.environ.get("CASAPATH")
pathname=pathname.split()[0]
datapath=pathname+"/data/demo/Images/imagetestimage.fits"
ia.fromfits(outfile='testimage.im', infile=datapath, overwrite=T) # 1
hdr = ia.summary()                                                # 2
ia.statistics()                                                   # 3
ia.close()                                                        # 4
print "Last example!  Exiting..."
exit()
#
"""
```

1. Convert (using `fromfits`) an image FITS disk file called *'imagetestimage.fits'* in the directory specified by the environment variable *'$CASAPATH'* to a CASA `image file` called *'testimage.im'*. The CASA `image file` is now associated with the default `image tool` **ia**.

2. This summarises (to the Logger) the basic header information in the image (name, masks, regions, brightness units, coordinates etc) and also stores it in a record named hdr.

3. This `tool function` evaluates basic statistics from the entire image.

4. This closes the Image `tool`, but does not delete its associated disk `image file`.

image-Tool.html

1.1.1 image - Tool

Operations on images
Requires: coordsys **Synopsis**

Description

Summary
Image tools provides access to CASA images. Currently only single precision floating point CASA images are supported by the Image `tool`. In the future, complex images will also be supported.
Image tools also provide direct (native) access to FITS and Miriad images. You can also convert these foreign formats to CASA format (for optimum processing speed).

Overview of Image `tool` functionality

- **Conversion -** There is functionality to interconvert between CASA images and FITS files. There is also native access to a FITS file:
  - fromfits - Convert a FITS image file to a CASA image
  - tofits - convert the image to a FITS file
  - image - native access to a FITS file
  - fromascii - Convert an ascii image file to a CASA image
  - fromarray - Convert an array into a CASA image
  - fromshape - Convert a shape into a CASA image
- **Analysis -**
  - subimage (function) - collapse image along specified axis, computing aggregate function of pixels along that axis
  - decompose - separate a complex image into individual components
  - deconvolvecomponentlist - deconvolve a Componentlist from the restoring beam
  - fft - FFT the image
  - findsources - Find strong point sources in sky
  - fitcomponents - Fit model components to an image.

- fitprofile - fit a 1-d profile with varying combinations of functional forms (see also the imageprofilefitter `tool`.
- histograms - compute histograms from the image
- insert - insert specified image into this image
- maxfit - Find maximum and do simple parabolic fit to sky
- modify - modify image by a model
- moments - compute moments from image
- regrid - regrid the image to the specified Coordinate System
- reorder - transpose the image (same as transpose())
- transpose - transpose the image (same as reorder())
- rotate - rotate the coordinate system and regrid the image to the rotated Coordinate System
- rebin - rebin an image by the specified binning factors
- statistics - compute statistics from the image
- twopointcorrelation - compute two point autocorrelation functions from the image
- subimage (function) - Create a (sub)image from a region of the image

- **Coordinates -** Manipulation of the coordinate system is handled through

  - coordmeasures - convert from pixel to world coordinate wrapped as Measures
  - coordsys - recover the Coordinate System into a Coordsys `tool`.
  - setcoordsys - set a new Coordinate System
  - topixel - convert from world coordinate to pixel coordinate
  - toworld - convert from pixel coordinate to world coordinate

  The coordsys `tool` provides more extensive coordinate system manipulation.

- **Filtering -** Images may be filtered via

  - convolve - Convolve image with an array or by another image
  - convolve2d - Convolve image by a 2D kernel
  - sepconvolve - Separable convolution
  - hanning - Hanning convolution along one axis

  In the future filtering other than convolution will be provided

- **Masks -** Masks may be manipulated via
  - calcmask - Image mask calculator
  - maskhandler - handle masks (set, copy, delete, recover names)
  - replacemaskedpixels - replace the values of pixels which are masked bad
  - set - set pixel and/or mask values with a scalar in a `region-of-interest` of the image
  - summary - lists the mask names
- **Pixel access -** The pixel and mask values for an image may be accessed and calculated with via
  - imagecalc - Create image tool with image calculator
  - image - Create an image tool from a CASA image
  - calc - Image pixel calculator
  - calcmask - Image mask calculator
  - getchunk - get the pixel values from a regular region of the image into an array
  - getregion - get pixels and mask from a `region-of-interest` of the image
  - getslice - get a 1-D slice from the image
  - pixelvalue - get image value for specified pixel
  - putchunk - put pixels from an array into a regular region of the image
  - putregion - put pixels and mask into a `region-of-interest` of the image
  - set - set pixel and/or mask values with a scalar in a `region-of-interest` of the image
- **Inquiry -** Functions to report basic information about the image are
  - boundingbox - find bounding box of a `region-of-interest`.
  - brightnessunit - Get image brightness unit
  - haslock - does this image have a lock set
  - history - recover/list history file
  - ispersistent - is the image persistent (on disk)
  - name - name of the `image file` this tool is attached to
  - restoringbeam - Get restoring beam
  - shape - the length of each axis in the image

  - summary - summarize basic information about the image
  - type - the type of this Image `tool`

- **Utility -** There is wide range of utility services available through the functions
  - adddegaxes - Add degenerate axes
  - addnoise - Add noise to the image
  - brightnessunit - Get image brightness unit
  - close - Close the `image tool` (but don't destroy it)
  - convertflux Convert flux density between peak and integral
  - close - close this `image tool`
  - haslock - does this image have a lock set
  - history - recover/list history file
  - imagefiles - Find the names of all image files in the given directory
  - imagetools - Find the names of all global image `tools`
  - is_image - Is this variable an Image `tool`
  - isopen - Is this Image `tool` open?
  - lock - acquire a lock on the image
  - makecomplex - make a complex image from two real images
  - miscinfo - recover miscellaneous information record
  - open - open a new `image file` with this image tool
  - rename - rename the `image file` associated with this Image `tool`
  - restoringbeam - Get restoring beam
  - remove - remove the `image file` associated with this Image `tool`
  - setbrightnessunit - Set image brightness unit
  - sethistory - set the history file
  - setmiscinfo - set the miscellaneous information record
  - setrestoringbeam - Set new restoring beam
  - unlock - release lock on this `image file`

- **Reshaping -** Images can be reshaped via
  - fromimage - Create a (sub)image from a region of a CASA image
  - subimage - Create a (sub)image from a region of the image
  - insert - insert specified image into this image
  - imageconcat - Concatenate CASA images

General

We refer to a CASA `image file` when we are referring to the actual data stored on disk. The name that you give a CASA `image file` is actually the name of a directory containing a collection of CASA tables which together constitute the `image file`. But you only need to refer to the directory name and you can think of it as one *logical* file.

Whenever we use the word "image", we are just using it in a generic sense. CASA images are manipulated with an Image tool associated with, or bound to, the actual `image file`. Note that some `image tools` don't have a disk file associated with them. These are called "virtual" images and are discussed below

When an image is stored on disk, it can, in principle, be stored in a variety of ways. For example, the image could be stored row by row; this is the way that most older generation packages store images. It makes for very fast row by row access, but very slow in other directions (e.g. extract all the profiles along the third axis of an image). A CASA `image file` is stored with what is called tiling. This means that small multi-dimensional chunks (a tile) are stored sequentially. It means that row by row access is a little slower, but access speed is essentially the same in all directions. This in turn means that you don't need to (and can't !) reorder images.

Here are some simple examples using image tools.

```
"""
#
print "\t----\t Intro Ex 1 \t----"
ia.maketestimage('zz',overwrite=true)# Make test image; writes disk file called 'zz'
print ia.summary()                   # Summarize (to logger)
print ia.statistics()                # Evaluate statistics over entire image
box = rg.box([10,10], [50,50])       # Make a pixel box region with regionmanager
im2 = ia.subimage('zz2', box, overwrite=true)  # Make a subimage  called 'zz2'
print im2.statistics()                     # Evaluate statistics
print "CLEANING UP OLD zz2.amp/zz2.phase IF THEY EXIST.  IGNORE WARNINGS!"
ia.removefile('zz2.amp')
ia.removefile('zz2.phase')
im2.fft(amp='zz2.amp',phase='zz2.phase')  # FFT subimage and store amp and phase
im2.done()      # Release tool resources - disk file unaffected
ia.close()      # DO NOT DONE DEFAULT IMAGE TOOL ia!!!
#
"""
```

Foreign Images

The Image `tool` also provides you with native access to some foreign image formats. Presently, these are FITS (Floar, Double, Short and Long are supported) and Miriad. This means that you don't have to convert the file to native CASA format in order to access the image. For example:

```
"""
#
print "\t----\t Intro Ex 2 \t----"
pathname=os.environ.get("CASAPATH") # Assumes environment variable is set
pathname=pathname.split()[0]
datapath1=pathname+"/data/demo/Images/imagetestimage.fits"
datapath2=pathname+"/data/demo/Images/test_image"
ia.open(datapath1)              # Access FITS image
#ia.open('im.mir')              # Access Miriad image (no image in repository)
ia.open(datapath2)              # Access casa image
#
#ims = ia.newimagefromimage(infile=datapath1, region=rg.quarter())
# rg.quarter() not implemented yet so has grabbed entire image
ims = ia.newimagefromimage(infile=datapath1)
innerquarter=rg.box([0.25,0.25],[0.75,0.75],frac=true)
subim = ims.subimage(region=innerquarter)
print ia.name()
print ims.name()
print subim.name()
ims.done()                      # done on-the-fly image tool
subim.done()                    # done on-the-fly image tool
ia.close()                      # close (not done) default image analysis tool
#
"""
```

```
Each of these Image tools has access to all the same \toolfunctions.
```

Where ever you see an argument in an Image `tool` function which is an input image disk file, that disk file can be a CASA, FITS, or Miriad image file.
There are some performance penalties that you should be aware of. Firstly, because CASA images are tiled (see above) you get the same access speed regardless of how you access the image. FITS and Miriad images are not tiled. This means that the performance for these Image tools will be poorer for certain operations. For example, extracting a profile along the third axis of an image, or re-ordering an image with the display library.
Secondly, for FITS images, masked values are indicated via "magic value". This means that the mask is worked out on the fly every time you access the image.
If you find performance is not good enough or you want a writable image, then use appropriate function (fromfits to convert to a native CASA image.

Virtual Images
We also have Image `tools` that are not associated one-to-one with disk files; these are called "virtual" images (see also the article in the AugustNewsLetter). For example, with the image calculator, imagecalc, one can create an expression which may contain many images. You can write the

result of the expression out to a disk `image file`, but if you wish, you can also just maintain the expression, evaluating it each time it is needed - nothing is ever written out to disk in this case. There are other Image functions like this (the documentation for each one explains what it does). The rules are:

- If you specify the `outfile` argument, then the image is always written to the specified disk `image file`.
- If you leave the `outfile` argument unset, then if possible, a virtual image will be created. Sometimes this virtual image will be an expression as in the example above (i.e. it references other images) or a temporary image in memory, or a temporary image on disk. (the summary function will list for you the type of image you have). When you destroy that Image tool, the virtual image will be destroyed as well.
- If you leave `outfile` unset, and the function cannot make a virtual image, it will create a disk file for you with a name of its choice (usually input plus function name).
- You can always write a virtual image to disk with the subimage `tool function`.

Coordinate Systems
An image contains a Coordinate System. A Coordsys `tool` is used to manipulate the Coordinate System. An Image `tool` allows you to recover the Coordinate System into a Coordsys `tool` through the coordsys function. You can set a new Coordinate System with the setcoordsys function.
You can do some direct coordinate conversion via the Image `tool` functions toworld, topixel, and coordmeasures. The actual work is done by a Coordsys `tool`, for which these Image `tool` functions are just wrappers.

Lattice Expression Language (LEL)
LEL allows you to manipulate expressions involving images. For example, add this image to that image, or multiply the miniumum value of that image by the square root of this image. The LEL syntax is quite rich and is described in detail in note 223.
LEL is accessed via the imagecalc and the calc `tool` functions. Here are some examples.

```
"""
#
print "\t----\t Intro Ex 3 \t----"
ia.maketestimage('zz', overwrite=true) # Make nonvirtual test image
ia.calc('zz + min(zz)')                # Make the minimum value zero
ia.close()
#
"""
```

In this example the Image `tool` is associated with the non-virtual disk file *zz*. This `image file` name is used in an LEL expression.

`Note that for image file names with special characters in them (like a dash for example), you should (double) escape those characters or put the file name in double quotes. E.g.`

```
"""
#
print "\t----\t Intro Ex 4 \t----"
ia.maketestimage("test-im", overwrite=true)
im1 = ia.imagecalc(pixels='test\\-im')  # Note double escape required
im2 = ia.imagecalc(pixels='"test-im"')
im1.done()
im2.done()
ia.close()
#
"""
```

Region-of-interest
A `region-of-interest` or simply, region, designates which pixels of the image you are interested in for some (generally) astrophysical reason. This complements the `pixel mask` (see below) which specifies which pixels are good or bad (for statistical reasons). `Regions-of-interest` are generated and manipulated with the Regionmanager tool.
Briefly, a `region-of-interest` may be either a simple shape such as a multi-dimensional box, or a 2-D polygon, or some compound combination of `regions-of-interest`. For example, a 2-D polygon defined in the X and Y axes extended along the Z axis, or perhaps a union or intersection of regions. See the Regionmanager documentation for more details on regions.
Regions are always supplied to `tool functions` via the `region` argument.

Pixel mask
A `pixel mask` specifies which pixels are to be considered good (value `T`) or bad (value `F`). For example, you may have imported a FITS file which has blanked pixels in it. These will be converted into `pixel mask` elements whose values are bad (`F`). Or you may have made an error analysis of an image and computed via a statistical test that certain pixels should be masked out for future analysis.
If there is no `pixel mask`, all pixels are considered good (if you retrieve the `pixel mask` when there is none, you will get an all good mask). Pixels for which the `pixel mask` value is bad are not used in computations (e.g. in the calculation of statistics, moments or convolution).

The image may contain zero, one, or more `pixel masks`. However, only one mask will be designated as the default mask. This is the `pixel mask` that is actually applied to the data. You can also indicate that none of the `pixel masks` are the default, so that effectively an all good `pixel mask` is applied. The function summary includes in its summary of the image the names of the masks (the first listed, if not in square brackets, is the default).
`Pixel masks` are handled with the function maskhandler. This allows you to find the names of `pixel masks`, delete them, copy them, nominate the default and so on. It is not used to change the value of `pixel masks`.
The functions with which you can change `pixel mask` values are putregion (put Boolean array), calcmask (put result of Boolean LEL expression), and set (put scalar Boolean).

The argument 'mask'
There is an argument, `mask`, which can be supplied to many functions. It is supplied with either a mask `region-of-interest` (generated via the function wmask) or a LEL Boolean expression string (the same string you would have supplied to the above Regionmanager function). Generally, one just supplies the expression string.
The LEL expression is simply used to generate a `pixel mask` which is then applied in addition to any default `pixel mask` in the image (a logical OR). For example

```
"""
#
print "\t----\t Intro Ex 5 \t----"
ia.maketestimage('zz', overwrite=true)
ia.statistics(mask='zz > 0')      # Only evaluate for positive values
ia.calcmask (mask='(2*zz) > 0')   # Create a new mask which is T (good)
                                  # when twice the image values are
                                  # positive, else F
ia.close()
#
"""
```

The `mask` expression must in general conform (shape and coordinates) with the image (i.e. that associated with the Image tool).
When `mask` is used with function calcmask, a persistent `pixel mask` is created and stored with the image. With all other functions, the `mask` argument operates as a transient (or On-The-Fly [OTF]) `pixel mask`. It can be very handy for analysing or displaying images with different masking criteria. Often I will refer to the "total input mask". This is the combination (logical OR) of the default `pixel mask` (if any) and the OTF mask (if any).

In the following example we open a Rotation Measure image. We then evaluate statistics and display it where only those pixels whose error in the Rotation Measure (`image file` *rmerr*) is less than the specified value are

shown; the others are masked. The nice thing is you can experiment with different `pixel masks` until you are satisfied, whereupon you might then make the `pixel mask` persistent with the calcmask function.

```
"""
#
print "\t----\t Intro Ex 6 \t----"
#myim = ia.newimagefromimage('rm')
#myim.statistics(mask='rmerr<10')
#myim.calcmask (mask='rmerr<20')     # Make persistent mask
#
"""
```

Finally, a subtlety that is worth explaining.

```
"""
#
print "\t----\t Intro Ex 7 \t----"
ia.maketestimage('zz', overwrite=true)
ia.statistics(mask='zz>0')                  # Mask of zz ignored
ia.statistics(mask='mask(zz) && zz>0')      # Mask of zz used
ia.close()
#
"""
```

In the first example, any default mask associated with the image {\sff zz} is ignored. Only the pixel values are looked at. In the second example, the mask of {\sff zz} is also taken into account via the LEL {\cf mask} function. That is, the transient output mask is T (good) only when the mask of {\sff zz} is T and the expression {\cf zz>0} is T.

A useful part of LEL to use with the `mask` argument is the `indexin` function. This enables the user to specify a mask based upon selected pixel coordinates or indices (specified 0-rel) rather than image values. For example

```
"""
#
print "\t----\t Intro Ex 8 \t----"
ia.fromshape(shape=[20])
print ia.getregion(mask='indexin(0, [4:9, 14, 18:19])',getmask=true)
```

```
#[False False False False True True True True True True False False False
# False True False False False True True]
ia.close()
#
"""
```

You can see the mask is good (T) for the specified indices along the specified axis. You can also pass in a premade variable for the specification if you like, viz.

```
"""
#
print "\t----\t Intro Ex 9 \t----"
ia.fromshape(shape=[20])
axis = "0"
sel = "[4:9, 14, 18:19]"
print ia.getregion(mask='indexin('+axis+','+sel+')',getmask=true)
#[False False False False True True True True True True False False False
# False True False False False True True]
ia.close()
#
"""
```

This capability is useful for fitting functions.

Pixel masks and Regions

Some comment about the combination of `pixel masks` and `regions-of-interest` is useful here. See the Regionmanager tool for basic information about `regions-of-interest` first.

Regions are provided to Image `tool functions` via the standard `region` function argument.

Consider a simple polygonal region. This `region-of-interest` is defined by a bounding box, the polygonal vertices, and a mask called a `region mask`. The `region mask` specifies whether a pixel within the bounding box is inside or outside the polygon. For a simple box `region-of-interest`, there is obviously no need for a `region mask`.

Now imagine that you wish to recover the `pixel mask` of an image from a polygonal `region-of-interest`. The mask is returned to you in regular Boolean array. Thus, the shape of the returned mask array reflects the bounding-box of the polygonal region. If the actual `pixel mask` that you apply is all good, then the retrieved mask would be good inside of the polygonal region and bad outside of it. If the actual `pixel mask` had some bad values in it as well, the retrieved mask would be bad outside of the polygonal region. Inside the polygonal region it would be bad if the `pixel mask` was bad. More simply put, the mask that you recover is just a logical "and" of the `pixel mask` and the `region mask`; if the `pixel mask` is T *and* the `region mask` is T then the retrieved mask is T (good), else it is F (bad).

Finally, note that if you use the `region` and `mask` (the OTF mask) arguments together then they operate as follows. The shape of the Boolean expression provided by `mask` must be the same shape as the image to which it is being applied. The `region` is applied equally to the image and the `mask` expression. For example

```
"""
#
print "\t----\t Intro Ex 10 \t----"
#rm1 = ia.newimagefromimage('rm')
#rm2 = ia.newimagefromimage('rmerr')
#rm1.shape()
#[128 128]
#rm2.shape()
#[128 128]
#r = rg.box([10,10], [50,50])
#rm1.statistics(region=r, mask='rmerr<10') # region applied to
                                            # 'rmerr' and 'rm'
#
"""
```

Methods

newimage	Construct a new image analysis tool using the specified image. (Also known as n
newimagefromfile	Construct a new image analysis tool using the specified image. (Also known as n
imagecalc	Perform mathematical calculations on an image or images.
collapse	Collapse an image along a specified axis, computing a specified aggregate functio
decimate	Remove planes from an image.
imageconcat	Construct a CASA image by concatenating images
fromarray	Construct a CASA image from a numerical (integer or float) array
fromascii	This function converts a pre-existing ascii file into a CASA image.
fromfits	Construct a CASA image by conversion from a FITS image file
fromimage	Construct a (sub)image from a region of a CASA image
fromshape	Construct an empty CASA image from a shape
maketestimage	Construct a CASA image from a test FITS file
adddegaxes	Add degenerate axes of the specified type to the image
addnoise	Add noise to the image
convolve	Convolve image with an array or another image
boundingbox	Get the bounding box of the specified region
boxcar	Convolve one axis of image with a boxcar kernel
brightnessunit	Get the image brightness unit
calc	Image calculator
calcmask	Image mask calculator
close	Close the image tool

continuumsub	Image plane continuum subtraction
convertflux	Convert peak intensity to/from flux density for a 2D Gaussian.
convolve2d	Convolve image by a 2D kernel
coordsys	Get the Coordinate System of the image
coordmeasures	Convert from pixel to world coordinate wrapped as Measures
decompose	Separate a complex image into individual components
deconvolvecomponentlist	Deconvolve a componentlist from the restoring beam
deconvolvefrombeam	Helper function to deconvolve the given source Gaussian from a beam Gaussian t
beamforconvolvedsize	Determine the size of the beam necessary to convolve with the given source to re
commonbeam	Determine a beam to which all beams in an image can be convolved.
remove	Delete the image file associated with this image tool
removefile	Delete an unattached image file from disk. Note: use remove() if the image file is
done	Destroy this image tool
fft	FFT the image
findsources	Find point sources in the sky
fitprofile	Fit gaussians and/or polynomials to a 1-dimensional profile.
fitcomponents	Fit 2-dimensional models to an image.
fromrecord	Generate an image from a record
getchunk	Get the pixel values from a regular region of the image into an array
getregion	Get pixels or mask from a region-of-interest of the image
getprofile	Get values and mask for a one dimensional profile along a specified image axis by
getslice	Get 1-D slice from the image
hanning	Convolve one axis of image with a Hanning kernel
haslock	Does this image have any locks set?
histograms	Compute histograms from the image
history	Recover and/or list the history file
insert	Insert specified image into this image
isopen	Is this Image `tool` open?
ispersistent	Is the image persistent?
lock	Acquire a lock on the image
makecomplex	Make a complex image
maskhandler	Handle pixel masks
miscinfo	Get the miscellaneous information record from an image
modify	Modify image with a model
maxfit	Find maximum and do parabolic fit in the sky
moments	Compute moments from an image
name	Name of the image file this tool is attached to
open	Open a new image file with this image tool
pad	Pad the perimeter of the direction plane with a number of pixels of specified valu
crop	Crop masked pixels from the perimeter of an image.
pixelvalue	Get value of image and mask at specified pixel coordinate
putchunk	Put pixels from an array into a regular region of the image
putregion	Put pixels and mask into a region-of-interest of the image
rebin	Rebin an image by the specified integer factors
regrid	regrid this image to the specified Coordinate System
transpose	Transpose the image.

rotate	rotate the direction coordinate axes attached to the image and regrid the image t
rotatebeam	rotate the image's beam(s) counterclockwise through the specified angle.
rename	Rename the image file associated with this image tool
replacemaskedpixels	replace the values of pixels which are masked bad
restoringbeam	Get the restoring beam(s).
sepconvolve	Separable convolution
set	Set pixel and/or mask values with a scalar in a region-of-interest of the image
setbrightnessunit	Set the image brightness unit
setcoordsys	Set new Coordinate System
sethistory	Set the history for an image
setmiscinfo	Set the miscellaneous information record for an image
shape	Length of each axis in the image
setrestoringbeam	Set the restoringbeam
statistics	Compute statistics from the image
twopointcorrelation	Compute two point correlation function from the image
subimage	Create a (sub)image from a region of the image
summary	Summarize basic information about the image
tofits	Convert the image to a FITS file
toASCII	Convert the image to an ASCII file
torecord	Return a record containg the image associated with this tool
type	Return the type of this tool
topixel	Convert from world to pixel coordinate
toworld	Convert from pixel to world coordinate
unlock	Release any lock on the image
newimagefromarray	Construct a CASA image from an array
newimagefromfits	Construct a CASA image by conversion from a FITS image file
newimagefromimage	Construct an on-the-fly image tool from a region of a CASA image file
newimagefromshape	Construct an empty CASA image from a shape
pbcor	Construct a primary beam corrected image from an image and a primary beam
pv	Construct a position-velocity image between two points in the direction plane.
makearray	Construct an initialized multi-dimensional array.
isconform	Returns true of the shape, coordinate system, and axes order of the specified ima

image.newimage.html

image.newimage - Function

1.1.1 Construct a new image analysis tool using the specified image. (Also known as newimagefromfile.)

Description

This method is identical to ia.newimagefromfile(). The description of how it works is in the online help for that method.

Arguments

Inputs		
infile	Input image file name	
	allowed:	string
	Default:	

Returns

image

---

image.newimagefromfile.html

image.newimagefromfile - Function

1.1.1 Construct a new image analysis tool using the specified image. (Also known as newimage.)

Description

This method returns an image analysis tool associated with the specified image. Constructing a image analysis tool in addition to the default ia tool allows the user to operate on multiple images without having to close one before opening another. All ia.newimagefrom*() methods share this functionality.
The parameter infile may refer to a CASA image, a Miriad image, or a FITS image. FITS images of types Float, Double, Long, and Short are supported.
When finished with the newly created tool, the user should close it to free up system resources (eg memory).
ia.newimage() is an alias for this method.

Arguments

Inputs		
infile	Input image file name	
	allowed:	string
	Default:	

Returns
image

Example

```
# This is one way to copy a FITS image into an already extant CASA image
# of the same shape (ia.subimage() is more effecient, but this example is
# meant to demonstrate ia.newimagefromfile()

# note that the ia tool is not attached to an image after the first command,
# the fitsimage tool is
```

```
fitsimage = ia.newimagefromfile("myimage.fits")
# now attach the target CASA image to the ia tool
ia.open("myimage.im")
# copy pixel values
ia.putchunk(fitsimage.getchunk())
# copy the coordinate system
ia.setcoordsys(fitsimage.coordsys().torecord())
# copy other miscellaneous things
ia.setbrightnessunit(fitsimage.getbrightnessunit())
ia.setmiscinfo(fitsimage.miscinfo())
# be sure to call done() on both tools to free up memory
ia.done()
fitsimage.done()
```

---

image.imagecalc.html

image.imagecalc - Function

1.1.1 Perform mathematical calculations on an image or images.

Description

This method is used to evaluate a mathematical expression involving existing images. It fully supports both float and complex valued images. The syntax of the expression supplied via the pixels parameter (in what is called the Lattice Expression Language, or LEL) is explained in detail in note 223. This is a rich mathematical language with allows all manner of mathematical operations to be applied to images.
Any image files embedded in the expression may be native CASA or FITS (but not yet Miriad) images.
If successful, this method always returns an image analysis tool that references the image resulting from the calculation. This returned tool should always be captured and closed as soon as the user is done with it to free up system resources (eg, memory). The image analysis tool on which the method is called (eg the ia tool when one runs ia.imagecalc()) remains unaltered, eg it still refers to the same image it did prior to the imagecalc() call.
Values of the returned tool are evaluated "on demand". That is, only when a method is run on the returned tool are the necessary values computed. And in fact, the values have to be reevaluated for each operation (method call). This means that there is a small performance hit for using the returned tool rather than the image written to disk and that none of the images which were used in the expression should be deleted while the returned tool is in use because they must be accessed for calculating the expression each time an operation of the returned tool is performed. These limitations do not apply to the ouput image if one is specified with the outfile parameter; it is a genuine CASA image with numerical values. If outfile is blank, no ouput image is written (although the resulting image can still be accessed via the returned image analysis tool as described below).
Normally you should just write the image, close the returned tool, and open the results image with the default ia tool and operate on it. If you are interested in conserving disk space, you don't need to keep the result of the calculation around for very long, and/or you are only going to do a small number of operations on the result image, should you set outfile="".
Note that when multiple image are used in the expression, there is no garauntee about which of those images will be used to create the metadata of the output image, unless imagemd is specified. If imagemd is specified, the following rules of metadata copying will be followed:

1. The pixel data type of the image specified by imagemd and the output image must be the same. 2. The metadata copied include the coordinate system (and so of course the dimensionality of the output image must correspond to the coordinate system to be copied), the image_info record (which contains things like the beam(s)), the misc_info record (should one exist in the image specified by imagemd), and the units. 3. If the output image is a spectral image, the brightness units are set to the empty string. 4. If the ouptut image is a polarization angle image, the brightness unit is set to "deg" and the stokes coordinate is set to have a single plane of type of Pangle.

Arguments

Inputs	
outfile	Output image file name. If blank the resulting image is not written, but it can still be accessed via the returned image analysis tool. allowed: string Default:
pixels	LEL expression. Must be specified. For example "myimage1.im + myimage2.im". allowed: string Default:
overwrite	Overwrite (unprompted) pre-existing output file? allowed: bool Default: false
imagemd	The image from which metadata should be copied. Default means no gaurantee from which image is used. allowed: string Default:

Returns
image

Example

```
"""
# Suppose aF and bF are images with single precision and we want
# to determine the result of the following expression:
# aF + min(float($\pi$, mean(bF))
```

```
#
# In this case, the images aF and bF do not need to have the same shapes and
# coordinates, because only the mean(bF) results in the mean of all pixel
# values in bF.  If aF has single precision pixel values, the resulting image
# will as well. This expression first computes the scalar value of the minimum
# of $\pi$ and the mean of the pixel values of bF. That scalar is then
# added to the value of each pixel in aF. In the code below, the
# result is written to image cF which can be accessed immediately
# via the returned image analysis tool captured in the variable myim.
# If the expression is masked, that mask will be copied to the new image.

# create images of different sizes to operate on
ia.fromshape('aF',[10,10],overwrite=true)
ia.fromshape('bF',[10,20,30],overwrite=true)
# close the ia tool to free up resources
ia.done()
# at each pixel in bF, take the minimum of that pixel value and pi and add
# the resulting value to the corresponding pixel in af
# note that only the subset of pixels in bF that correspond to those in aF
# are used; the resulting image has the same size as the smaller image, aF,
# used in the input
myim = ia.imagecalc(outfile='cF', pixels='aF + min(float(pi()), mean(bF))',
                overwrite=true)
# confirm the resulting image has the same size as aF, should be [10, 10]
myim.shape()
# close the myim tool to free up system resources
myim.done()
"""
```

Example

```
"""
# The following example shows the use of the two min() LEL functions. One takes a
# single argument and will return a scalar representing the minimum pixel value
# of that entire image. The other takes two arguments (either an image and a
# scalar or two images of conforming shapes) and returns an image for which
# the minimum has been calculated on a pixel by pixel basis for the input
# image(s).

# create an image to operate on
ia.fromshape('aF',[10,10],overwrite=true)
```

```
# give it interesting values
ia.addnoise()
# free up system resources
ia.done()
# do the calculation and write results to image cF
myim=ia.imagecalc('cF', 'min(aF, (min(aF)+max(aF))/2)', overwrite=true)
# do whatever stuff you want with myim and the close it to free
# up system resources
myim.done()
"""
```

Example

```
"""
# Here's an example of a more complicated function. Currently
# ia.fromshape() only creates real-valued images so the real()
# function is not particularly exciting in this case but illustrates
# possibilities. Trigonometric functions such as sin() assume the
# pixel values are in radians.
ia.fromshape('aD',[10,10],overwrite=true)
ia.addnoise()
ia.fromshape('aF',[10,10],overwrite=true)
ia.addnoise()
ia.fromshape('bF',[10,10],overwrite=true)
ia.addnoise()
ia.fromshape('aC',[10,10],overwrite=true)
ia.addnoise()
ia.done()
myim = ia.imagecalc('eF', 'sin(aD)+(aF*2)+min(bF)+real(aC)', overwrite=true)
myim.done()
"""
```

---

image.collapse.html

image.collapse - Function

1.1.1 Collapse an image along a specified axis, computing a specified aggregate function of pixels along that axis.

Description

This method collapses an image along a specified axis or set of axes of length N pixels to a single pixel on each specified axis. Both float valued and complex valued images are supported. It computes a user-specified aggregate function for pixel values along the specified axes, and places those values in the single remaining plane of those axes in the output image. The method returns an image analysis tool containing the newly-created collapsed image. Valid choices of aggregate functions are: 'flux' (see below for constraitns), 'max', 'mean', 'median', 'min', 'rms', 'stdev', 'sum' and 'variance'. Minimal unique matching is supported for the function parameter (e.g. function = 'r' will compute the rms of the pixel values, 'med' will compute the median, etc.).
If one specifies function='flux', the following constraints must be true:
1. The image must have a direction coordinate, 2. The image must have at least one beam, 3. The specified axes must be exactly the direction coordinate axes, 4. Only one of the non-directional axes may be non-degenerate, 5. The iamge brightness unit must be conformant with x*yJy/beam, where x is an optional unit (such as km/s for moments images) and y is an optional SI prefix.
Axes may be specified as a single integer or an array of integers indicating the zero-based axes along which to collapse the image. Axes may also be specified as a single or array of strings which minimally and uniquely match (ignoring case) world axis names in the image (e.g. 'dec' for collapsing along the declination axis or ['ri', 'd'] for collapsing along both the right ascension and declination axes).
If outfile is not specified (or contains only whitespace characters), no image is written but the collapsed image is still accessible via the image analysis tool this method always returns (which references the collapsed image). If the returned object is not wanted, it should still be captured and destroyed via its done() method. If this is not done, there is no guarantee as to when the Python garbage collector will delete it. If the returned object is wanted, it should still be deleted as soon as possible for the same reasons, e.g.
collapsed_image = ia.collapse(...)

```
# do things (or not) with the collapsed_image and when finished working with the object, do
```

collapsed_image.done()

The reference pixel of the collapsed axis is set to 0 and its reference value is set to the mean of the the first and last values of that axis in the specified region of the input image. The reference value is the world coordinate value of the reference pixel. For instance, if an axis to be collapsed were to be the frequency axis, in the collapsed image, the reference value would be the mean value of the frequency range spanned, and would be stored in pixel 0.
If the input image has per plane beams, the beam at the origin of the subimage determined by the selected region is arbitrarily made the global beam of the output image. In general, the user should understand the pitfalls of collapsing images with multiple beams (i.e. that employing an aggregate function on pixels with varying beam sizes more often than not leads to ill-defined results). Convolution to a common beam is not performed automatically as part of the preprocessing before the actual rebinning occurs. In such cases, therefore, the user should probably first convolve the input image with a common restoring beam so that each plane has the same resolution, and/or use imsmooth to smooth the data to have the same beam.

Arguments

Inputs	
function	Aggregate function to apply. This can be set one of flux, max, mean, median, min, rms, stdev, sum, variance. Must be specified. allowed: string Default:
axes	Zero-based axis number (specified as a list or integer) along which to collapse the specified image. Default value is 0. allowed: any Default: variant 0
outfile	Output image file name. If left blank (the default), no image is written but a new image tool referencing the collapsed image is returned. allowed: string Default:
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
box	Rectangular region to select in direction plane. See "help par.box" for details. Default is to use the entire direction plane. allowed: string Default:
chans	Channels to use. See "help par.chans" for details. Channels must be contiguous. Default is to use all channels. allowed: string Default:
stokes	Stokes planes to use. See "help par.stokes" for details. Planes specified must be contiguous. Default is to use all Stokes planes. allowed: string Default:
mask	Mask to use. See help par.mask for more details. Default setting is none. allowed: string Default:
overwrite	Overwrite (unprompted) pre-existing output file? Ignored if "outfile" is left blank. allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default value is False. allowed: bool Default: false

Returns
image

Example

```
"""
# myimage.im is a 512x512x128x4 (ra,dec,freq,stokes) image
ia.open("myimage.im")
# collapse a subimage of it along its spectral axis avoiding the 8 edge
# channels at each end of the band, computing the mean value of the pixels
# resulting image is 256x256x1x4 in size.
collapsed = ia.collapse(outfile="collapse_spec_mean.im", function="mean", axes=2, box="127,1
# manipulate collapsed
collapsed.done()

"""
```

---

image.decimate.html

image.decimate - Function

1.1.1 Remove planes from an image.

Description

This application removes planes along the specified axis of an image. It supports both float valued and complex valued images. The factor parameter represents the factor by which to reduce the number of planes.
The method parameter represents how to calculate the pixel values of the output image. A value of method="copy" means that every factorth plane of the selected region in the input image will be directly copied to the corresponding plane in the output image. So, if one wanted to copy every third spectral plane in the input image to the output image, one would specify factor=3 and method="copy". If the selected region along the specified axis had 11 planes, then there would be 4 output planes which would map to planes 0, 3, 6, and 9 of the specified region of input image. A value of method="mean" indicates that each of factor number of planes in the range starting at each factorth plane should be averaged to produce the corresponding output plane. So, if one specified factor=3 and method="mean" along an axis of the selected region of the input image which had 11 pixels, the corresponding axis in the output image would have three pixels and the pixel values for each of those output planes would corresponding to averaging along that axis planes 0-2, 3-5, and 6-8 of the selected region of the input image. Note that the remaining planes, 9 and 10, in the selected region of the input image would be ignored because the last interval must have exactly factor number of planes in order to be included in the output image.
The coordinate system of the output image takes into account the decimation; that is, along the decimated axis, the increment of the output image is factor times that of the input image, and the reference pixel of the output image is located at pixel 1/factor times the reference pixel in the input image.
This method returns an image analysis tool which references the output image. If this tool is not desired, one should capture it anyway and then close() it immediately to free up resources.
Images with multiple beams are not supported; please convolve a multi-beam image to a single resolution before running this application.

Arguments

Inputs	
outfile	Output image file name. If empty, a persistent image is not created. allowed: string Default:
axis	Axis along which to remove planes. allowed: int Default: 0
factor	Reduce number of planes by this factor. allowed: int Default: 1
method	Method to use for calculating pixel values of output. Supported values are "copy" or "mean". allowed: string Default: copy
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask for more details. Default setting is none. allowed: string Default:
overwrite	Overwrite (unprompted) pre-existing output file? Ignored if "outfile" is left blank. allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default value is False. allowed: bool Default: false

Returns
image

Example

```
# Copy verbatim every 5th plane of axis 2 of the input image
ia.open("myim.im")
decimated = ia.decimate("dec1.im", axis=2, factor=5, method="copy")
```

```
# do stuff with decimated and then close it
decimated.close()

# Decimte by averaging every 7 planes of the input image along axis 2
decimated = ia.decimate("dec2.im", axis=2, factor=7, method="mean")
# do stuff with decimated and then close it
decimated.close()
```

---

image.imageconcat.html

image.imageconcat - Function

1.1.1 Construct a CASA image by concatenating images

Description

This function is used to concatenate two or more input CASA images into one output image. For example, if you have two image cubes which are contiguous along one axis (say a spectral axis) and you would like to glue them together along this axis, then this function is the appropriate thing to use.
The axis parameter is used to specify which zero-based axis the images should be concatenated along. A negative value indicates that the spectral axis should be used. If a negative value is given but there is no spectral axis, an exception will be thrown. The zero-based order of the axes of an image can be determined from ia.coordsys().names().
If successful, this method will return an image analysis tool referencing the concatenated image. Even if it is not wanted, the returned tool should be captured and closed as soon as the user is finished with it to free up system resources (eg memory).
If `outfile` is given, the image is written to the specified disk file. If `outfile` is unset, the on-the-fly Image `tool` created by the function actually references all of the input files. So if you deleted any of the input image disk files, it would render this `tool` useless. When you destroy this tool (with the done function) the reference connections are broken.
The input and output images must be of the same dimensionality. Therefore, if you wish to concatenate 2-D images into a 3-D image, the 2-D images must have a third axis (of length unity) so that the output image coordinates are known along the concatenation axis.
The input images are concatenated in the order in which they are listed unless the reorder parameter is set to True. If True, the images are reordered if necessary so that the world coordinate values along the selected axis monotonically increase or decrease. The direction of the increment is determined by the first listed image. If reorder=True, the world coordinate ranges of the images along the selected axis are not permitted to overlap, and the signs of the increments for this axis in all images must be the same. If reorder=False, the coordinate system of the first listed image is used as the coordinate system for the output image. If reorder=True, the coordinate system of the first image in the list of the reordered images is used as the coordinate system of the output image. Setting reorder=True can be especially useful if the infiles are specified using a wildcard character(s).

If relax=False, the input images are checked to see that they are contiguous along the concatenation axis and an error is generated if they are not. In addition, the coordinate descriptors (e.g. reference pixel, reference value etc) for the non-concatenation axes must be the same or an error will result.
The input disk image files may be in native CASA, FITS, or Miriad formats.
The contiguous criterion and coordinate descriptor equality criteria can be relaxed by setting `relax=T` whereupon only warnings will be issued. Dimension and shape must still be the same though. When the concatenation axis is not contiguous (but still monotonically increasing or decreasing) and `relax=T`, a tabular coordinate will be used to correctly describe the axis. But be aware that it means adjacent pixels are not regularly spaced. However, functions like toworld and topixel will correctly interconvert world and pixel coordinates.
In giving the input image names, the `infiles` argument can be a single string if you wild card it with standard shell symbols. For example, `infiles='cena_???.*'`, where the "?" represents one character and "*" any number of characters.
Otherwise, you must input a vector of strings such as `infiles="cena1 cena2 cena3"`. An input such as `infiles='files1,file2'` will be interpreted as one string naming one file and you will get an error. The reason for this is that although the latter could be parsed to extract two file names by recognizing comma delimiters, it is not possible because an expression such as `infiles='cena.{a,b}'` (meaning files of name "cena.a" and "cena.b") would confuse such parsing (you would get two files of name *cena.*{*a* and *b*}.
You can look at the coordinate system of the output image using the ia.summary() tool method to ensure it's correct.
The argument `tempclose` is, by default, True. This means that all internal reference copies of the input images are kept closed until they are needed. Then they are opened temporarily and then closed again. This enables you to effectively concatenate as many images as you like without encountering any operating system open file number limits. However, it comes at some performance loss, because opening and closing all those files takes time. If you are concatenating a smallish number of files, you might use `tempclose=F`. This will leave all internal reference copies permanently open, but performance, if you don't hit the file limit, will be better.
This method requires multiple images which are specified with the infiles parameter. Therefore calling ia.open() is not necessary, although calling imageconcat() using an already open image analysis tool will work and the state of that tool (eg the image it references) will not be changed.

Arguments

Inputs	
outfile	Output image file name. Default is unset. allowed: string Default:
infiles	List of input CASA image files to concatenate; wild cards accepted. Default is empty string. allowed: any Default: variant
axis	Concatenation pixel axis. Use ia.coordsys().names() to get a list of axes. A negative value means use the spectral axis if there is one, if not an exception is thrown. allowed: int Default: -1
relax	Relax constraints that axis coordinate descriptors match allowed: bool Default: false
tempclose	Keep all lattices closed until needed allowed: bool Default: true
overwrite	Overwrite (unprompted) pre-existing output file? allowed: bool Default: false
reorder	Automatically reorder the images if necessary. allowed: bool Default: false

Returns
image

Example

```
"""
# Create three images to concatenate together.
ia.fromshape('im.1',[10,10,10],overwrite=T)
ia.fromshape('im.2',[10,10,10],overwrite=T)
ia.fromshape('im.3',[10,10,10],overwrite=T)
ia.done()
# now concatenate.
# The three images have the same shape along the axes not to be
```

```
# concatenated as they must. relax=T means that the contiguity
# constraint along the concatenated axis is not imposed (if it were
# the call would fail because the spectral axes of the input images
# are not contiguous).
bigim = ia.imageconcat(outfile='bigimage', infiles='im.1 im.2 im.3',
                       axis=2, relax=T, tempclose=F, overwrite=T)
# be sure to call done() on the return tool to free up system resources.
bigim.done()
"""
```

Example

```
"""
#
# All images whose file names begin with {\sff im.} that reside in
# the current directory are concatenated along the spectral axis if
# there is one. All image coordinate descriptors must match.  If any
# input image does not have a spectral axis an error will
# result.  Because an oufile is not specified, the returned image analysis
# tool captured in the variable named bigim just references the input images;
# this call does not create a persistent result image.
bigim = ia.imageconcat(infiles="im.*",relax=T)
bigim.done()
"""
```

---

image.fromarray.html

image.fromarray - Function

1.1.1 Construct a CASA image from a numerical (integer or float) array

Description

This function converts a numerical (integer or float) numpy array of any size and dimensionality into a CASA image. It will create both float and complex valued images.
The image analysis tool on which this method is called will reference the created image; if this tool referenced another image before this call, that image will no longer be referenced by the tool after the creation of the new image. If you would rather have a new image analysis tool returned, keeping the one on which this method is called unaltered, use newimagefromarray() instead. If `outfile` is given, the image is written to disk, if not, the image tool on which this method was called will reference a temporary image (either in memory or on disk, depending on its size) that will be deleted when the tool is closed. Float valued images are produced from real-valued arrays. Complex-valued images are produced from complex-valued arrays.
The coordinate system, provided as a coordsys `tool`converted to a record is optional. If you provide it, it must have the same number of dimensions as the pixels array (see also coordsys). Call the naxes() method on the coordinate system tool to see how many dimensions the coordinate system has. A coordinate system can be created from scratch using the coordinate system (cs) tool and methods therein, but often users prefer to use a coordinate system from an already existing image. This can be gotten using ia.coordsys() which returns a coordinate system tool. A torecord() call on that tool will result in a python dictionary describing the coordinate system which is the necessary format for the csys input parameter of ia.fromarray().
If csys is not specified, a default coordinate system is created. If `linear=F` (the default) the created coordinate system will have standard RA/DEC/Stokes/Spectral Coordinate axes depending upon the shape of the `pixels` array (Stokes axis must be no longer than 4 pixels and you may find the spectral axis preceding the Stokes axis if say, `shape=[64,64,32,4]`). Extra dimensions are given linear coordinates. If `linear=T`, then all the resulting coordinates are linear with the axes represent lengths. In this case each axis will have a value of 0.0 at its center pixel. The increment of each axis will be 1.0 km.
The method returns True if creation of the image was successful, False otherwise, so you can check programmatically if the image creation was successful.

Arguments

Inputs		
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	
pixels	Numeric array	
	allowed:	any
	Default:	variant
csys	Coordinate System. Default is unset.	
	allowed:	record
	Default:	
linear	Make a linear Coordinate System if csys not given	
	allowed:	bool
	Default:	false
overwrite	Overwrite (unprompted) pre-existing output file?	
	allowed:	bool
	Default:	false
log	Write image creation messages to logger	
	allowed:	bool
	Default:	true

Returns
bool

Example

```
"""
# make an image with a default RA/Dec/Stokes/Frequency coordinate system
# having all pixels set to 2.5.
ary = ia.makearray(v=2.5, shape=[64, 64, 4, 128])
# the ia tool does not need to reference an image in this case (ie open()
# need not have been called), if it does reference another image, that reference
# will be lost and replaced with a reference to the newly created image.
res = ia.fromarray(outfile='test.data', pixels=ary, overwrite=true)
if res:
    # perform operations on the newly created image if desired and make sure
    # to close it when done to free up system resources (eg memory)
    ia.shape()
```

```
ia.done()
"""
```

Example

```
"""

# create an image using the coordinate system from another image
ia.open("myexistingimage.im")

mycs = ia.coordsys()
# the number of dimensions in the array and the coordinate system must
# be the same. For this example to work, mycs.naxes() must return 4.
ia.done()
ary = ia.makearray(v=2.5, shape=[64, 64, 4, 128])
res = ia.fromarray(pixels=ary, csys=mycs.torecord())
mycs.done()
if (res):
    # do things with the newly created temporary image before closing it
    ia.shape()
ia.done()

"""
```

---

image.fromascii.html

image.fromascii - Function

1.1.1 This function converts a pre-existing ascii file into a CASA image.

Description

This function is used to create a CASA image from a pre-existing ASCII file. You might want to use this if you just want to create a quick image to use to see what various image analysis methods do. The image analysis tool on which the method is called will always reference the created image if this method is successful. Thus, calling open() on that tool is not necessary, but if the tool is already open, referencing another image, that reference will be silently destroyed and replaced with a reference to the image created by fromascii(). If `outfile` is given, the image is also written to the specified disk file. If `outfile` is unset, the image analysis tool on which this method was called references a temporary image. This temporary image may be in memory or on disk, depending on its size. When you close the image analysis `tool` (with the close function) the temporary image is deleted.
You must specify the shape of the image. The image must be stored in the ascii file. If the shape of the image having N axes is to be [s_0, s_1, s_2, ..., s_(N-1)], where s_i is the integral number of pixels along axis number i, the file must have (s_1 * s_2 * ... * s_(N-1)) rows and each row must have s_0 numerical values delimited by the value specified by the sep parameter, or spaces if sep is not specified. Pixel locations are incremented by row number in such a way that the second axis changes fastest. As an example, say we want to create an image with shape=[3,4,2]. There must be 4*2 = 8 rows in the ascii file, and each row must have 3 space-delimited numerical values. The first row represents values of pixels [0, 0, 0], [1, 0, 0], and [2, 0, 0]. The second row represents values of pixels [0, 1, 0], [1, 1, 0], and [2, 1, 0]. The fifth row represents values of pixels [0, 0, 1], [1, 0, 1], and [2, 0, 1]. The sixth values of pixels [0, 1, 1], [1, 1, 1], and [2, 1, 1,]. And so on.
To further illustrate, say this is our file:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
When read with ia.fromascii(), ary = ia.getchunk() would return the following array: ary[0, 0, 0] = 1 ary[1, 0, 0] = 2 ary[2, 0, 0] = 3 ary[0, 1, 0] = 4 ary[1, 1, 0] = 5 ary[2, 1, 0] = 6 ary[0, 2, 0] = 7 ary[1, 2, 0] = 8 ary[2, 2, 0] = 9 ary[0, 3, 0] = 10 ary[1, 3, 0] = 11 ary[2, 3, 0] = 12 ary[0, 0, 1] = 13 ary[1, 0, 1] = 14 ary[2, 0, 1] = 15 ary[0, 1, 1] = 16 ary[1, 1, 1] = 17 ary[2, 1, 1] = 18 ary[0, 2, 1] = 19 ary[1, 2, 1] = 20 ary[2, 2, 1] = 21 ary[0, 3, 1] = 22 ary[1, 3, 1] = 23 ary[2, 3, 1] = 24

The coordinate system, provided as a coordsys `tool` converted to a record with coordsys torecord, is optional. If you provide it, it must be dimensionally consistent with the pixels array you give (see also coordsys).
If you don't provide the coordinate system, a default coordinate system is made for you. If `linear=F` (the default) then it is a standard RA/DEC/Stokes/Spectral Coordinate System depending exactly upon the shape of the `pixels` array (Stokes axis must be no longer than 4 pixels and you may find the spectral axis coming out before the Stokes axis if say, `shape=[64,64,32,4]`). Extra dimensions are given linear coordinates. If `linear=T`, then all axes are linear in the resulting coordinate system.

Arguments

Inputs		
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	
infile	Input ascii disk file name. Must be specified..	
	allowed:	string
	Default:	
shape	Shape of image. Must be specified.	
	allowed:	intArray
	Default:	-1
sep	Separator in ascii file. Default is space character.	
	allowed:	string
	Default:	:
csys	Coordinate System record from coordsys torecord(). Default is unset.	
	allowed:	record
	Default:	
linear	Make a linear Coordinate System if csys not given	
	allowed:	bool
	Default:	false
overwrite	Overwrite (unprompted) pre-existing output file?	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
"""
# say the above file presented above is stored in a file
# named image.txt
ia.fromascii(outfile="myimage.im", infile='image.txt', shape=[3,4,2], overwrite=true)
# should return [3, 4, 2]
ia.shape()
# call other ia methods then close the tool
ia.done()
"""
```

---

image.fromfits.html

image.fromfits - Function

1.1.1 Construct a CASA image by conversion from a FITS image file

Description

This function is used to convert a FITS disk image file (Float, Double, Short, Long are supported) to an CASA `image file`. If `outfile` is given, the image is written to the specified disk file. If `outfile` is unset, the Image `tool` is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you close the Image `tool` (with the close function) this temporary image is deleted.
This function reads from the FITS primary array (when the image is at the beginning of the FITS file; `whichhdu=0`), or an image extension (when the image is elsewhere in the FITS file, `whichhdu` $>$ `0`).
By default, any blanked pixels will be converted to a mask value which is false, and a pixel value that is NaN. If you set `zeroblanks=T` then the pixel value will be zero rather than NaN. The mask will still be set to false. See the function replacemaskedpixels if you need to replace masked pixel values after you have created the image.

Arguments

Inputs		
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	
infile	Input FITS disk file name. Must be specified.	
	allowed:	string
	Default:	
whichrep	If this FITS file contains multiple coordinate representations, which one should we read (0-based)	
	allowed:	int
	Default:	0
whichhdu	If this FITS file contains multiple images, which one should we read (0-based).	
	allowed:	int
	Default:	0
zeroblanks	If there are blanked pixels, set them to zero instead of NaN	
	allowed:	bool
	Default:	false
overwrite	Overwrite (unprompted) pre-existing output file?	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
"""
#
print "\t----\t fromfits Ex 1 \t----"
datapath=pathname+'/data/demo/Images/imagetestimage.fits'
ia.fromfits('./myimage', datapath, overwrite=true)
print ia.summary()
s = ia.miscinfo()
print s.keys()  #  prints any unrecognized field names
ia.close()
#
"""
```

The FITS image is converted to a \casa\ \imagefile\ and access is
provided via the default \imagetool\ called {\stf ia}. Any FITS
header keywords which were not recognized or used are put in the
miscellaneous information bucket accessible with the miscinfo function. In
the example we list the names of the fields in this record.

---

image.fromimage.html

image.fromimage - Function

1.1.1 Construct a (sub)image from a region of a CASA image

Description

This function applies a `region-of-interest` to an `image file`, creates a new `image file` containing the (sub)image, and associates the `image tool` with it. The input image file may be in native CASA, FITS, or Miriad format. Look here for more information on foreign images.
If `outfile` is given, the (sub)image is written to the specified disk file.
If `outfile` is unset, the Image `tool` actually references the input image file. So if you deleted the input image disk file, it would render this `tool` useless. When you close this `tool` (with the close function) the reference connection is broken.
Sometimes it is useful to drop axes of length one (degenerate axes). Use the `dropdeg` argument if you want to do this.
The output mask is the combination (logical OR) of the default input `pixel mask` (if any) and the OTF mask. Any other input `pixel masks` will not be copied. Use function maskhandler if you need to copy other masks too.
See also the subimage function.

Arguments

Inputs		
outfile	Output sub-image file name. Default is unset.	
	allowed:	string
	Default:	
infile	Input image file name. Must be specified.	
	allowed:	string
	Default:	
region	Region selection. See "help par.region" for details. Default is to use the full image.	
	allowed:	any
	Default:	variant
mask	Mask to use. See help par.mask. Default is none.	
	allowed:	any
	Default:	variant
dropdeg	Drop degenerate axes	
	allowed:	bool
	Default:	false
overwrite	Overwrite (unprompted) pre-existing output file?	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
"""
#
print "\t----\t fromimage Ex 1 \t----"
innerquarter = rg.box([0.25,0.25],[0.75,0.75],frac=true)
ia.close()
ia.fromimage(outfile='image.small', infile='test.data', region=innerquarter, overwrite=true)
ia.close()
#
"""

The specified \region\ takes a quarter by area of the first two axes
of the image, and all pixels of other axes.
```

image.fromshape.html

image.fromshape - Function

1.1.1 Construct an empty CASA image from a shape

Description

This function creates a CASA `image file` with the specified shape. All the pixel values in the image are set to 0. One may create either an image with float valued pixels (type='f') or a complex valued image (type='c').
If `outfile` is given, the image is written to the specified disk file. If `outfile` is unset, the Image `tool` is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you close the Image `tool` (with the close function) this temporary image is deleted.
The Coordinate System, provided as a Coordsys `tool`, is optional. If you provide it, it must be dimensionally consistent with the shape that you specify (see also coordsys).
If you don't provide the Coordinate System, a default Coordinate System is made for you. If `linear=F` (the default) then it is a standard RA/DEC/Stokes/Spectral Coordinate System depending exactly upon the shape (Stokes axis must be no longer than 4 pixels and you may find the spectral axis coming out before the Stokes axis if say, `shape=[64,64,32,4]`). Extra dimensions are given linear coordinates. If `linear=T` then you get a linear Coordinate System.
The method returns True if successful, False otherwise.

Arguments

Inputs		
outfile	Name of output image file. Default is unset.	
	allowed:	string
	Default:	
shape	Shape of image. Must be specified.	
	allowed:	intArray
	Default:	0
csys	Coordinate System. Default is unset.	
	allowed:	record
	Default:	
linear	Make a linear Coordinate System if csys not given?	
	allowed:	bool
	Default:	false
overwrite	Overwrite (unprompted) pre-existing output file?	
	allowed:	bool
	Default:	false
log	Write image creation messages to logger	
	allowed:	bool
	Default:	true
type	Type of image. 'f' means Float, 'c' means complex.	
	allowed:	string
	Default:	f

Returns
bool

Example

```
"""
#
print "\t----\t fromshape Ex 1 \t----"
ia.fromshape('test2.data', [64,64,128], overwrite=true)
mycs = ia.coordsys(axes=[0,2])
ia.close()
ia.fromshape(shape=[10, 20], csys=mycs.torecord())
mycs.done()
ia.close()
#
"""
```

The first example creates a zero-filled \imagefile\ named {\sff
test.data} of shape [64,64,128]. If you examine the header with {\stff
ia.summary()} you will see the RA/DEC/Spectral coordinate information.
In the second example, a Coordinate System describing the first and third axes
of the image {\sff test.data} is created and used to create a 2D temporary image.

---

image.maketestimage.html

image.maketestimage - Function

1.1.1 Construct a CASA image from a test FITS file

Description

This function converts a FITS file resident in the CASA system into a CASA image.
If outfile is given, the image is written to the specified disk file. If outfile is unset, the Image tool is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you close the Image tool (with the close function) this temporary image is deleted.

Arguments

Inputs		
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	
overwrite	Overwrite (unprompted) pre-existing output file?	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
"""
#
print "\t----\t maketestimage Ex 1 \t----"
ia.maketestimage()     # make virtual image
ia.close()
ia.maketestimage('tmp', overwrite=true)
ia.close()             # close to unlock disk image
```

```
#
"""
```

---

image.adddegaxes.html

image.adddegaxes - Function

1.1.1 Add degenerate axes of the specified type to the image

Description

This method adds degenerate axes (i.e. axes of length 1) of the specified type. Sometimes this can be useful although you will generally need to modify the coordinate system of the added axis to give it the coordinate you want (do this with the Coordsys `tool`). This method supports both float and complex valued images.
You specify which type of axes you want to add. You can't add an axis type that already exists in the image. For the Stokes axis, the allowed value (a string such as I, Q, XX, RR) can be found in the Coordsys newcoordsys function documentation.
If `outfile` is given, the image is written to the specified disk file. If `outfile` is unset, the on-the-fly Image `tool` returned by the function is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the generated Image `tool` (with the done function) this temporary image is deleted.

Arguments

Inputs		
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	
direction	Add direction axes?	
	allowed:	bool
	Default:	false
spectral	Add spectral axis?	
	allowed:	bool
	Default:	false
stokes	Add Stokes axis? Default is empty string.	
	allowed:	string
	Default:	
linear	Add linear axis?	
	allowed:	bool
	Default:	false
tabular	Add tabular axis?	
	allowed:	bool
	Default:	false
overwrite	Overwrite (unprompted) pre-existing output file?	
	allowed:	bool
	Default:	false
silent	Skip silently existing axes?	
	allowed:	bool
	Default:	false

Returns
image

Example

```
"""
#
print "\t----\t adddegaxes Ex 1 \t----"
ia.maketestimage()
print ia.shape()
#[113L, 76L]
mycs=ia.coordsys()
print mycs.axiscoordinatetypes()
#['Direction', 'Direction']
```

```
mycs.done()
im2 = ia.adddegaxes(spectral=T)
print im2.shape()
#[113L, 76L, 1L]
mycs=im2.coordsys()
print mycs.axiscoordinatetypes()
['Direction', 'Direction', 'Spectral']
mycs.done()
im3 = im2.adddegaxes(stokes='Q')
print im3.shape()
#[113L, 76L, 1L, 1L]
mycs = im3.coordsys()
print mycs.axiscoordinatetypes()
#['Direction', 'Direction', 'Spectral', 'Stokes']
mycs.done()
im2.done()
im3.done()
ia.close()
#
"""
```

In this example, all the images are virtual (temporary images).

---

image.addnoise.html

image.addnoise - Function

1.1.1 Add noise to the image

Description

This function adds noise to the image. You may zero the image first before the noise is added if you wish.
The noise can be drawn from one of many distributions.
For each distribution, you must supply the type via the `type` argument (minimum match is active) and parameters via the `pars` argument. Briefly:

- binomial – the binomial distribution models successfully drawing items from a pool. Specify two parameters, n and p, respectively. n is the number of items in the pool, and p, is the probability of each item being successfully drawn. It is required that $n > 0$ and $0 \leq p \leq 1$.
- discreteuniform – models a uniform random variable over the closed interval. Specify two parameters, the low and high values, respectively. The low parameter is the lowest possible return value and the high parameter is the highest. It is required that $low < high$.
- erlang – Specify two parameters, the mean and variance, respectively. It is required that the mean is non-zero and the variance is positive.
- geometric – Specify one parameter, the probability. It is required that $0 \leq probability < 1$.
- hypergeometric – Specify two parameters, the mean and the variance. It is required that the variance is positive and that the mean is non-zero and not bigger than the square-root of the variance.
- normal – Specify two parameters, the mean and the variance. It is required that the variance is positive.
- lognormal – Specify two parameters, the mean and the variance. It is required that the supplied variance is positive and that the mean is non-zero.
- negativeexponential – Supply one parameter, the mean.
- poisson – Specify one parameter, the mean. It is required that the mean is non-negative.

- uniform – Model a uniform random variable over a closed interval. Specify two parameters, the low and high values. The low parameter is the lowest possible return value and the high parameter can never be returned. It is required that $low < high$.
- weibull – Specify two parameters, alpha and beta. It is required that the alpha parameter is not zero.

  The random number generator seeds may be specified as an array of integers. Only the first two values are used. If none or a single value is provided, the necessary remaining value(s) are generated based on the current time, using the algorithm

  ```
  seedBase = 1e7*MJD
  seed[1] = (Int)seedBase;
  # and if seed[0] is also not supplied
  seed[0] = (Int)((1e7*(seedBase - seed[1])))
  ```

  where MJD is the Modidfied Julian Day.

Arguments

Inputs		
type	Type of distribution, normal	
	allowed:	string
	Default:	normal
pars	Parameters of distribution	
	allowed:	doubleArray
	Default:	0.0 1.0
region	Region selection. See "help par.region" for details. Default is to use the full image.	
	allowed:	any
	Default:	variant
zero	Zero image first?	
	allowed:	bool
	Default:	false
seeds	Seeds to use for the random number generator. If not specified, seeds are created based on the current time.	
	allowed:	intArray
	Default:	

Returns
bool

Example

```
"""
ia.maketestimage()
ia.addnoise(type='normal', pars=[0.5, 1], zero=T, seeds=[1,2])
ia.statistics()
ia.close()
"""
```

A test image is created, zeroed, and noise of mean 0.5 and variance 1
from a normal distribution added. Specifying the same combination of seeds
ensures the same random number sequence is created each time addnoise is called.
To have different sequences created during the same casapy session, use the default
value (which is an empty list).

---

image.convolve.html

image.convolve - Function

1.1.1 Convolve image with an array or another image

Description

This function performs Fourier-based convolution of an `image file` by the given kernel.
If `outfile` is given, the image is written to the specified disk file. If `outfile` is left unset, the on-the-fly Image `tool` generated by this function is associated with a temporary image. This temporary image may be stored in memory or on disk, depending on its size. When the user destroys the generated Image `tool` (with the done function) this temporary image is deleted.
The kernel is provided as a multi-dimensional array or as the filename of a disk-based `image file`. The provided kernel can have fewer dimensions than the image being convolved. In this case, it will be padded with degenerate axes. An error will result if the kernel has more dimensions than the image. No additional scaling of the kernel is provided yet.
The scaling of the output image is determined by the argument `scale`. If this is left unset, then the kernel is normalized to unit sum. If `scale` is not left unset, then the convolution kernel will be scaled (multiplied) by this value. Masked pixels will be assigned the value 0.0 before convolution.
The output mask is the combination (logical OR) of the default input `pixel mask` (if any) and the OTF mask. Any other input `pixel masks` will not be copied. The function maskhandler should be used if there is a need to copy other masks too.
See also the other convolution functions:
convolve2d, sepconvolve and hanning.

Arguments

Inputs	
outfile	Output image file name. Default is unset. allowed: string Default:
kernel	Convolution kernel - An array or an image filename. Must be specified by the user. allowed: any Default: variant
scale	Scale factor. The default behavior is to autoscale (specified with -1.0). allowed: double Default: -1.0
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: variant Default: variant
mask	Mask to use. See help par.mask for more details. The default setting is none. allowed: any Default: variant
overwrite	Overwrite (unprompted) the pre-existing output file? allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch for more details. allowed: bool Default: false

Returns
image

Example

```
"""
#
print "\t----\t convolve Ex 1 \t----"
# This example presupposes the existence of an input image file, testdata, and a kernel imag
```

```
# Open the input image file:
ia.open(infile='testdata')
# Set up a region to be operated upon (in this case, the whole image):
r1 = rg.box()
# Perform the convolution:
im2 = ia.convolve (outfile = 'convout', overwrite = true, region = r1, kernel = 'kerneldata'
ia.close()
im2.done()

#
print "\t----\t convolve Ex 2 \t----"
# This example uses an array as the convolution kernel, and presupposes the existence of an
ia.open(infile='testdata')
# Next, create a Python array of some kind to use as a convolution kernel. For example:
from numpy import arange
kernelarray = arange(10)**3
# Set up a region to be operated upon (in this case, the whole image):
r1 = rg.box()
# Perform the convolution:
im3 = ia.convolve (outfile = 'convout2', overwrite = true, region = r1, kernel = kernelarray
ia.close()
im3.done()
#
"""
```

---

image.boundingbox.html

image.boundingbox - Function

1.1.1 Get the bounding box of the specified region

Description

This function finds the bounding box of a region of interest when it is applied to a particular image. Both float and complex valued images are supported. It is returned in a record which has fields `'blc', 'trc', 'inc', 'bbShape', 'regionShape', 'imageShape', 'blcf'` and `'trcf'` containing the bottom-left corner, the top-right corner (in absolute image pixel coordinates), the increment (stride) of the region, the shape of the boundingbox, the shape of the region, the shape of the image, the blc in formatted absolute world coordinates and the trc in formatted absolute world coordinates, respectively. Note that the shape of the bounding box will be different from the shape of the region if a non-unit stride (increment) is involved (see the example below). Note that the integer size of the elements in blc, trc, inc, regionShape, bbShape, and imageShape are 32 bits, even on a 64 bit machine. This means that, on 64 bit machines, you may have to convert them to 64 bit ints using eg numpy.int64, before being able to use them as direct input to other methods such as ia.getchunk().

Arguments

Inputs	
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant

Returns
record

Example

```
"""
#
print "\t----\t boundingbox Ex 1 \t----"
ia.maketestimage()                              # Create image tool
x=['3pix','6pix','9pix','6pix','5pix','5pix','3pix'] # X vector in abs pixels
y=['3pix','4pix','7pix','9pix','7pix','5pix','5pix'] # Y vector in abs pixels
mycs = ia.coordsys()
r1=rg.wpolygon(x=x,y=y,csys=mycs.torecord()) # Create polygonal world region
mycs.done()
bb = ia.boundingbox(r1)                         # Find bounding box
print bb
#{'regionShape': array([7, 7]), 'trc': array([9, 9]),
# 'imageShape': array([113, 76]),
# 'blcf': '00:00:27.733, -00.06.48.000',
# 'trcf': '00:00:24.533, -00.05.36.000', 'bbShape': array([7, 7]),
# 'blc': array([3, 3]), 'inc': array([1, 1])}
ia.close()
#
"""
```

Example

```
"""
#
print "\t----\t boundingbox Ex 2 \t----"
ia.maketestimage()
b = rg.box([10,10],[20,20],[2,3])
print ia.boundingbox(b)
#{'regionShape': array([6, 4]), 'trc': array([20, 19]),
# 'imageShape': array([113,  76]),
# 'blcf': '00:00:24.000, -00.05.24.000',
# 'trcf': '00:00:18.667, -00.03.36.000', 'bbShape': array([11, 10]),
# 'blc': array([10, 10]), 'inc': array([2, 3])}

ia.close()
#
"""
```

In this example we see the difference between bbShape and regionShape because of the increment (stride). See also that the trc is modified by the increment.

---

image.boxcar.html

image.boxcar - Function

1.1.1 Convolve one axis of image with a boxcar kernel

Description

This application performs boxcar convolution of one axis of an image defined by
z[i] = (y[i] + y[i+i] + ... + y[i+w])/w
where z[i] is the value at pixel i in the box car smoothed image, y[k] is the pixel value of the input image at pixel k, and w is a postivie integer representing the width of the boxcar in pixels. Both float and complex valued images are supported. The length of the axis along which the convolution is to occur must be at least w pixels in the selected region, unless decimation using the mean function is chosen in which case the axis length must be at least 2*w (see below). Masked pixel values are set to zero prior to convolution. All nondefault pixel masks are ignored during the calculation. The convolution is done in the image domain (i.e., not with an FFT).
If drop=False (no decimation), the length of the output axis will be equal to the length of the input axis - w + 1. The pixel mask, ORed with the OTF mask if specified, is copied from the selected region of the input image to the output image. Thus for example, if the selected region in the input image has six planes along the convolution axis, if the specified boxcar width is 2, and if the pixel values, which are all unmasked, on a slice along this axis are [1, 2, 5, 10, 17, 26], then the corresponding output slice will be of length five and the output pixel values will be [1.5, 3.5, 7.5, 13.5, 21.5].
If drop=True and dmethod="copy", the output image is the image calculated if drop=True, except that only every wth plane is kept. Both the pixel and mask values of these planes are copied directly to the output image, without further processing. Thus for example, if the selected region in the input image has six planes along the convolution axis, the boxcar width is chosen to be 2, and if the pixel values, which are all unmasked, on a slice along this axis are [1, 2, 5, 10, 17, 26], the corresponding output pixel values will be [1.5, 7.5, 21.5].
If drop=True and dmethod="mean", first the image described in the drop=False case is calculated. Then, the ith plane of the output image is calculated by averaging the i*w to the (i+1)*w-1 planes of this intermediate image. Thus, for example, if the selected region in the input image has six planes along the convolution axis, the boxcar width is chosen to be 2, and if the pixel values, which are all unmasked, on a slice along this axis are [1, 2, 5, 10, 17, 26], then the corresponding output pixel values will be [2.5, 10.5]. Any pixels at the end of the plane of the intermediate image that do not fall into a

complete bin of width w are ignored. Masked values are taken into consideration when forming this average, so if one of the values is masked, it is not used in the average. If at least one of the values in the intermediate image bin is not masked, the corresponding output pixel will not be masked.
The smoothed image is written to disk with name `outfile`, if specified. If not, no image is written but the image is still accessible via the returned image analysis tool (see below).
This method always returns an image analysis tool which is attached to the smoothed image. This tool should always be captured and closed after any desired manipulations have been done. Closing the tool frees up system resources (eg memory), eg,

```
smoothedim = ia.boxcar(...)
# do things (or not) with smoothedim
...
# close the returned tool promptly upon finishing with it.
smoothedim.done()
```

Arguments

Inputs	
outfile	Output image file name. Default is none. allowed: string Default:
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
axis	Zero-based axis along which to convolve. ia.coordsys().names() gives the order of the axes in the image. Less than 0 means use the spectral axis if there is one, if not an exception is thrown. allowed: int Default: -1
width	Width of the boxcar in pixels. allowed: int Default: 2
drop	Drop every nth pixel on output, where n is the width of the boxcar? allowed: bool Default: true
dmethod	If drop=True, method to use in plane decimation. "copy": direct copy of every second plane, "m(ean)": average planes n*i through n*(i+1) - 1 (inclusive) in the smoothed, non-decimated image to form plane i in the output image. allowed: string Default: copy
overwrite	Overwrite (unprompted) pre-existing output file? allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false

Returns
image

Example

```
ia.open("mynonsmoothed.im")
# smooth the spectral axis by 3 pixels, say it's axis 2 and only
# write every other pixel
boxcar = ia.boxcar(outfile="myboxcarsmoothed.im", axis=2, drop=True,
width=3, dmethod="c" overwrite=True)
# done with input
ia.done()
# do something with the output image, get statistics say
stats = boxcar.statistics()
# close the result image
boxcar.done()
```

---

image.brightnessunit.html

image.brightnessunit - Function

1.1.1 Get the image brightness unit

Description

This function gets the image brightness unit. Both float and complex valued images are supported.

Arguments

Returns
string

Example

```
"""
#
print "\t----\t brightnessunit Ex 1 \t----"
ia.maketestimage()
print ia.brightnessunit()
#Jy/beam
ia.close()
#
"""
```

---

image.calc.html

image.calc - Function

1.1.1 Image calculator

Description

This function is used to evaluate a mathematical expression involving CASA images, assigning the result to the current (already existing) image. Both float and complex valued images are supported, although the image which results from the calculation must have the same type of pixel values as the image already attached to the tool. That is, one cannot create a complex valued image using this method if the associated ia tool is currently attached to a float valued image. It complements the imagecalc function which returns a newly constructed on-the-fly image tool. See note 223 which describes the the syntax and functionality in detail.
If the expression, supplied via the `pixels` argument, is not a scalar, the shapes and coordinates of the image and expression must conform.
If the image (that associated with the tool) has a `pixel mask`, then only pixels for which that mask is good will be changed. See the function maskhandler for managing image `pixel masks`.
Note that when multiple image are used in the expression, there is no garauntee about which of those images will be used to create the header of the output image. Therefore, one may have to modify the output header as needed if the input headers differ.
See the related functions set and putregion.

Arguments

Inputs		
pixels	LEL expression	
	allowed:	string
	Default:	
verbose	Emit possibly useful messages.	
	allowed:	bool
	Default:	True

Returns
bool

Example

```
"""
#
print "\t----\t calc Ex 1 \t----"
ia.maketestimage('aF', overwrite=true)
ia.calc('min(aF, (min(aF)+max(aF))/2)')
ia.calc('1.0')
ia.close()
#
"""
```

The first example shows that there are 2 {\cf min} functions. One with a single argument returning the minimum value of that image. The other with 2 arguments returning an image containing ‘‘aF’’ data clipped at the value of the 2nd argument. The second example sets all good pixels to unity.

```
"""
#
print "\t----\t calc Ex 2 \t----"
ia.maketestimage('aD', overwrite=true)       # create some
ia.close()
ia.maketestimage('aF', overwrite=true)       # image files
ia.close()
ia.maketestimage('bF', overwrite=true)       # for use
ia.close()
ia.maketestimage('aC', overwrite=true)       # in
ia.close()
ia.maketestimage()
ia.calc('sin(aD)+(aF*2)+min(bF)+real(aC)')   # the example
ia.close()
#
"""
```

This shows a mixed type expression. The real part of the complex image ‘‘aC’’ is used in an expression that otherwise uses Float type.

---

image.calcmask.html

image.calcmask - Function

1.1.1 Image mask calculator

Description

This method is used to create a new `pixel mask` via a Boolean LEL expression. This gives you much more scope than the simple set and putregion functions. Both float and complex valued images are supported.
See http://casa.nrao.edu/aips2_docs/notes/223/index.shtml which describes the the syntax and functionality of LEL in detail. Also in this document is a description of ways to escape image names that contain certain non-alphanumeric characters so they are compatible with LEL syntax.
If the expression is not a scalar, the shapes and coordinates of the image and expression must conform. If the expression is a scalar then the entire `pixel mask` will be set to that value.
By default (argument `name`) the name of a new `pixel mask` is made up for you. However, if you specify a `pixel mask` name (use function summary or maskhandler to see the mask names) then it is used. If the `pixel mask` already exists, it is overwritten.
You can specify whether the new `pixel mask` should be the default mask or not. By default, it is made the default `pixel mask` !

Arguments

Inputs		
mask	Mask to use. See help par.mask. Default is none.	
	allowed:	string
	Default:	
name	Mask name. Default is auto new name.	
	allowed:	string
	Default:	
asdefault	Make specified mask the default mask?	
	allowed:	bool
	Default:	true

Returns
bool

Example

```
"""
#
print "\t----\t calcmask Ex 1 \t----"
ia.maketestimage('zz', overwrite=true)
subim = ia.subimage()                # Make "another" image
ia.calcmask('T')                     # Specify 'True' mask as a string
ia.calcmask('zz>0')                  # Mask of zz ignored
ia.calcmask('mask(zz) && zz>0')      # Mask of zz included
ia.calcmask(subim.name(true)+'>min('+subim.name(true)+')') # Use tool names
ia.calcmask('zz>min(zz:nomask)')  # Mask of zz not used in scalar function
subim.done()
ia.close()
#
"""


The first calcmask example is the equivalent of {\cf
ia.set(pixelmask=1)}.  It sets the entire mask to True.

The second example creates a new \pixelmask\ which is True when
the pixel values in image {\sff zz} are greater than 0.

Now for some subtlety.  Read carefully !  Any LEL expression can be
thought of as having a value and a mask.  Usually the value is Float and
the mask Boolean.  In this case, because the expression is Boolean
itself, the value is also Boolean.  The expression mask would just be
the mask of {\sff zz}.  Now what {\stfaf calcmask} does is create a mask
from the expression value (which is Boolean) and discards the expression
mask.  Therefore, the resulting mask is independent of any mask
that {\sff zz} might have.

If you wish the mask of the expression be honoured as well,
then you can do as in the third example.   It says the output \pixelmask\
will be True if the current \pixelmask\ of {\sff zz} is True and the expression
value is True.

The fourth example is like the second, except that we use the pixel
values associated with the on-the-fly {\stf subim} Image tool  disk file.  Note one further
subtlety here.  When the scalar function {\cf min} evaluates a value
from {\cf subim.name()}, which in this case is just {\cf zz}, the default
mask of {\cf subim.name()} {\it will} be used.  All the scalar
```

functions look at the mask. If you didn't want the mask to be used
you can use the special {\cf :nomask} syntax shown in the final
example.

---

image.close.html

image.close - Function

1.1.1 Close the image tool

Description

This function closes the `image tool`. This means that it detaches the tool from its `image file` (flushing all the changes first). The `image tool` is "null" after this change (it is not destroyed) and calling any `tool function` other than open will result in an error.

Arguments

Returns
bool

Example

```
"""
#
print "\t----\t close Ex 1 \t----"
ia.maketestimage('myimage',overwrite=true)  # First create an image and attach the image too
ia.close()              # The Image tool is detached from the image using the close tool.
print "!!!EXPECT ERROR HERE!!!"
ia.summary()            # The image is not open, so attempting to display summary informatio
ia.open('myimage')      # The image tool is reattached to the image using the open tool.
ia.summary()            # No error - the summary information is now displayed correctly.
ia.close()              # The Image tool is detached from the image again, using the close t
#
"""
```

image.continuumsub.html

image.continuumsub - Function

1.1.1 Image plane continuum subtraction

Description

This function packages the relevant image tool functionality for simple specification and application of image plane continuum subtraction. All that is required of the input image is that it have a non-degenerate spectral axis. The user specifies region, the region of the input image over which continuum subtraction is desired (otherwise the whole image will be treated); channels, the subset of channels on the spectral axis to use in the continuum estimation, specified as a vector; fitorder, the polynomial order to use in the estimation. Optionally, output line and continuum images may be written by specifying outline and outcont, respectively. If outline is not specified, a virtual image tool is all that is produced. If outcont is not specified, the output continuum image will be written in 'continuumsub.im'. Note that the pol parameter is no longer supported; one should use the region parameter if polarization selection is desired, in conformance with other ia tool methods.

Arguments

Inputs		
outline	Output line image filename. Default is unset.	
	allowed:	string
	Default:	
outcont	Output continuum image filename	
	allowed:	string
	Default:	continuumsub.im
region	Region selection. See "help par.region" for details. Default is to use the full image.	
	allowed:	any
	Default:	variant
channels	Channels to use for continuum estimation. Default is all.	
	allowed:	intArray
	Default:	-1
pol	THIS PARAMETER IS NO LONGER SUPPORTED. USE THE region PARAMETER TO CHOOSE WHICH POLARIZATIONS YOU WOULD LIKE TO PROCESS	
	allowed:	string
	Default:	
fitorder	Polynomial order for continuum estimation	
	allowed:	int
	Default:	0
overwrite	Auto-overwrite output files if they exist?	
	allowed:	bool
	Default:	false

Returns
image

Example

```
Fit a second order polynomial (fitorder=2) to channels 3-8 and 54-60 (python's
range function includes the lower limit and excludes the upper limit)
to an RA x Dec x Frequency x Stokes cube.

ia.open("myimage.im")
```

```
# select stokes plane 1 on which to perform the fit, as well
# as a box of pixels with blc=25,25 and trc=75,75 in the direction
# plane, and channels 0 to 100. This will be the portion of the cube
# from which the fit is subtracted
reg = rb.box(blc=[25, 25, 0, 1], trc=[75, 75, 100, 1])
csub = ia.continuumsub(region=reg, channels=range(3,9)+range(54,61), fitorder=2)

# do stuff with original image (ia) and csub image tools as necessary and
# finally close them
ia.done()
csub.done()
```

---

image.convertflux.html

image.convertflux - Function

1.1.1 Convert peak intensity to/from flux density for a 2D Gaussian.

Description

This function interconverts between peak intensity and flux density for a Gaussian component. The image must hold a restoring beam.

Arguments

Inputs	
value	Flux density to convert. Must be specified. allowed: any Default: variant 0Jy/beam
major	Major axis of component. Must be specified. allowed: any Default: variant 1arcsec
minor	Minor axis of component. Must be specified. allowed: any Default: variant 1arcsec
type	Type of component. String from Gaussian, Disk. allowed: string Default: Gaussian
topeak	Convert to peak or integral flux desnity allowed: bool Default: true
channel	Channel to use if and only if image has per plane beams. allowed: int Default: -1
polarization	Zero-based polarization number to use for beam if and only if image has per plane beams. allowed: int Default: -1

Returns

record

Example

```
"""
#
print "\t----\t convertflux Ex 1 \t----"
ia.maketestimage('in.im', overwrite=true);
p1 = qa.quantity('1mJy/beam')
i1 = ia.convertflux(p1, major='30arcsec', minor='10arcsec', topeak=F);
p2 = ia.convertflux(i1, major='30arcsec', minor='10arcsec', topeak=T)
print 'peak, integral, peak = ', p1, i1, p2
#peak, integral, peak =  {'value': 1.0, 'unit': 'mJy/beam'}
#                        {'value': 0.00016396129551656742, 'unit': 'Jy'}
#                        {'value': 0.0010000000000000002, 'unit': 'Jy/beam'}

ia.close()
#
"""
```

image.convolve2d.html

image.convolve2d - Function

1.1.1 Convolve image by a 2D kernel

Description

This function performs Fourier-based convolution of an `image file` using the provided 2D kernel.
If `outfile` is left unset, the image is written to the specified disk file. If `outfile` is not given, the newly constructed on-the-fly Image `tool` is associated with a temporary image. This temporary image may be stored in memory or on disk, depending on its size. When the user destroys the on-the-fly Image `tool` (with the done function) this temporary image is deleted.
The user specifies which 2 pixel axes of the image are to be convolved via the `axes` argument. The pixels must be square or an error will result.
The user specifies the type of convolution kernel with `type` (minimum match is supported); currently only `'gaussian'` is available.
The user specifies the parameters of the convolution kernel via the arguments `major`, `minor`, and `pa`. These arguments can be specified in one of three ways:

- Quantity - for example `major=qa.quantity(1, 'arcsec')` Note that you pixel units can be used, viz. `major=qa.quantity(1, 'pix')`, see below.
- String - for example `minor='1km'` (i.e. one that the Quanta quantity function accepts).
- Numeric - for example `major=10`. In this case, the units of `major` and `minor` are assumed to be in pixels. Using pixel units allows the user to convolve unlike axes (see one of the provided example for this use case). For the position angle, units of degrees are assumed.

The interpretation of `major` and `minor` depends upon the kernel type.

- Gaussian - `major` and `minor` are the Full Width at Half Maximum (FWHM) of the major and minor axes of the Gaussian.

The position angle is measured North through East when a plane holding a celestial coordinate (the usual astronomical convention) is convolved. For other axis/coordinate combinations, a positive position angle is measured from +x to +y in the absolute pixel coordinate frame (x is the first axis that is specified, with argument `axes`).

In the case of a Gaussian, the `beam` parameter offers an alternate way of describing the convolving Gaussian. If used, neither `major`, stfaf minor, nor `pa` can be specified. The `beam` parameter must have exactly three fields: "major", "minor", and "pa" (or "positionangle"). This is, not coincidentally, the record format for the output of ia.restoringbeam().
The scaling of the output image is determined by the argument `scale`. If this is left unset then autoscaling will be invoked.
If the user is not convolving the sky, then autoscaling means that the convolution kernel will be normalized to have unit volume so as to conserve flux.
If the user is convolving the sky, then there are two cases for which autoscaling is useful:
Firstly, if the input image units are Jy/pixel, then the output image will have units of Jy/beam and be appropriately scaled. In addition, the restoring beam of the output image will be the same as the convolution kernel.
Secondly,if the input image units are Jy/beam, then the output image will also have units of Jy/beam and be appropriately scaled. In addition, the restoring beam of the output image will be the convolution of the input image restoring beam and the convolution kernel. In the case of an image with per-plane beams, for each plane, the kernel is convolved with the appropriate beam and the result is associated with that plane in the output image.
If the user sets a value for `scale`, then the convolution kernel will be scaled by this value. Note that it has peak of unity before the application of this scale factor.
If the units on the original image include Jy/beam, the units on the output image will be rescaled by the ratio of the input and output beams as well as rescaling by the area of convolution kernel.
If the units on the original image include K, then only the image convolution kernel rescaling is done.
If targetres=True and type="gaussian" and the input image has a restoring beam, this method will interpret the values of major, minor, and pa as the resolution of the final image and will calculate the parameters of the Gaussian to use in the convolution so that this target resolution is achieved.
Masked pixels will be assigned the value 0.0 before convolution. The output mask is the combination (logical OR) of the default input `pixel mask` (if any) and the OTF mask. Any other input `pixel masks` will not be copied. The function maskhandler can be used if there is a need to copy other masks too.
See also the other convolution functions:
convolve, hanning, and sepconvolve.

Arguments

Inputs	
outfile	Output image file name. The default value is unset. allowed: string Default:
axes	Axes to convolve. The default setting is [0,1]. allowed: intArray Default: 01
type	Type of convolution kernel to be used. allowed: string Default: gaussian
major	Major axis, Quantity, string, numeric (e.g. 10arcsec, 20pix, 3km, etc.). This must be specified by the user. allowed: any Default: variant 0deg
minor	Minor axis, Quantity, string, numeric (e.g. 10arcsec, 20pix, 3km, etc.). This must be specified by the user. allowed: any Default: variant 0deg
pa	Position Angle, Quantity, string, numeric (e.g. 10deg). The default value is 0deg. allowed: any Default: variant 0deg
scale	Scale factor. The default setting (-1) is to autoscale. allowed: double Default: -1
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default:
mask	Mask to use. See help par.mask for more details. The default option is none. allowed: any Default: variant
overwrite	Overwrite (unprompted) the pre-existing output file? allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch. allowed: bool Default: false
targetres	If True and type="gaussian", major, minor, and pa are interpreted as the image resolution that the user wants to achieve. allowed: bool Default: false
beam	Alternate way of describing a Gaussian. Must have fields "major", "minor", and "pa" (or "positionangle") allowed: record Default:

Returns
image

Example

```
"""
#
print "\t----\t convolve2d Ex 1 \t----"
ia.maketestimage('xy',overwrite=true)         # Create a simple RA/DEC test image
# Convolve axes 0 and 1 of the test image with a 20x10-arcsec, 45-degree Gaussian:
im2 = ia.convolve2d(outfile='xy.con', axes=[0,1], type='gauss',
                    major='20arcsec', minor='10arcsec', pa='45deg',
                    overwrite=true);
# Clean up, by destroying the im2 tool and close the image tool:
im2.done()
ia.close()
#

ia.fromarray(outfile='xypf', pixels=ia.makearray(0, [64, 64, 4, 64]),
             overwrite=true)         # Create a simple RA/DEC/Pol/Freq test dataset
print "!!!EXPECT WARNING REGARDING INVALID SPATIAL RESTORING BEAM!!!"
# Convolve axes 0 and 3 of the test dataset with a 20x10-pixel, 45-degree Gaussian:
im2 = ia.convolve2d(outfile='xypf.con', axes=[0,3], type='gauss',
                    major='20pix', minor='10pix', pa='45deg',
                    overwrite=true);
# Note that pixel units must be used in the above because axes 0 and 3 are unlike.
# Clean up, by destroying the im2 tool and close the image tool:
im2.done()
ia.close()
#
"""
```

image.coordsys.html

image.coordsys - Function

1.1.1 Get the Coordinate System of the image

Description

This function returns the Coordinate System of an image in a **Coordsys** tool. Both float and complex valued images are supported.
By default, the Coordinate System describes all of the axes in the image. If you desire, you can select a subset of the axes, thus reducing the dimensionality of the Coordinate System. This may be useful if you are supplying a Coordinate System to the functions fromarray or fromshape.

Arguments

Inputs		
axes	Axes to which the Coordinate System pertains. Default is all axes.	
	allowed:	intArray
	Default:	-1

Returns
coordsys

Example

```
"""
#
print "\t----\t coordsys Ex 1 \t----"
ia.maketestimage('hcn',overwrite=true)
ia.summary()
mycs = ia.coordsys([0,1])
imshape = ia.shape()
ia.fromshape(outfile='test', shape=imshape, csys=mycs.torecord(), overwrite=true)
ia.summary()
```

```
mycs.done()
ia.close()
#
"""
```

In this example, we create a Coordinate System pertaining to the first two axes of the image and then we create a new (empty) 2D image with this Coordinate System using the {\cf fromshape} function.

---

image.coordmeasures.html

image.coordmeasures - Function

1.1.1 Convert from pixel to world coordinate wrapped as Measures

Description

You can use this function to get the world coordinates for a specified absolute pixel coordinate in the image. You specify a pixel coordinate (0-rel) for each axis in the image.
If you supply fewer pixel values then there are axes in the image, your value will be padded out with the reference pixel for the missing axes. Excess values will be ignored.
The parameters dframe and sframe allow one to specify to which reference frame the direction and spectral measures, respectively, should be converted. These values are case-insensitive. "native" means use the native reference frame of the coordinate in question. "cl" means use the conversion layer frame if one exists (if not, the native frame will be used).
The world coordinate is returned as a record of measures. This function is just a wrapper for the Coordsys tool toworld function (invoked with argument `format='m'`). Please see its documentation for discussion about the formatting and meaning of the measures.
This Image `tool` function adds two additional fields to the return record. The `mask` field contains the value of the image `pixel mask` at the specified position. It is either T (pixel is good) or F (pixel is masked as bad or the specified position was off the image).
The `intensity` field contains the value of the image (at the nearest pixel to that given) and its units. This is actually stored as a Quantity. This field does not exist if the specified pixel coordinate is off the image.

Arguments

Inputs	
pixel	Absolute pixel coordinate. Default is reference pixel. allowed: doubleArray Default: -1
dframe	Direction reference frame to which to convert the direction data. Case insensitive. "cl" means use the conversion layer, if present, of the image direction coordinate. "native" means use the native native direction frame of the image. Other examples are "J2000", "B1950", "GALACTIC", etc. allowed: string Default: cl
sframe	Spectral reference frame to which to convert the spectral data. Case insensitive. "cl" means use the conversion layer, if present, of the image spectral coordinate. "native" means use the native spectral reference frame of the image. Other examples are "LSRK", "CMB", "LGROUP", etc. allowed: string Default: cl

Returns
record

Example

```
"""
#
print "\t----\t coordmeasures Ex 1 \t----"
ia.maketestimage('myimage',overwrite=true)
s = ia.shape()
for i in range(len(s)):
  s[i] = 0.5*s[i]
meas = ia.coordmeasures(s)
print meas.keys()                    # Get names of fields in record
#['intensity', 'mask', 'measure']
print meas['intensity']
#{'value': 1.39924156665802, 'unit': 'Jy/beam'}
print meas['measure']['direction']
#{'type': 'direction',
```

```
# 'm1': {'value': 5.817764003289323e-05, 'unit': 'rad'},
# 'm0': {'value': -5.8177644130875234e-05, 'unit': 'rad'}, 'refer': 'J2000'}
dir = meas['measure']['direction']  # Get direction coordinate
me.doframe(me.observatory('ATCA'))  # Set location on earth
me.doframe(me.epoch('utc','16jun1999/12:30:20'))  # Set epoch
azel = me.measure(dir,'azel')       # Convert to azimuth/elevation
print 'az,el=', qa.angle(azel['m0']), qa.angle(azel['m1'])  # Format nicely
#az,el= +105.15.47 -024.22.57
meas2=ia.coordmeasures()            # defaults to reference pixel
print meas2['intensity']
#{'value': 2.5064315795898438, 'unit': 'Jy/beam'}
print meas2['measure']['direction']
#{'type': 'direction',
# 'm1': {'value': 0.0, 'unit': 'rad'},
# 'm0': {'value': 0.0, 'unit': 'rad'}, 'refer': 'J2000'}
dir = meas2['measure']['direction'] # Get direction coordinate
me.doframe(me.observatory('ATCA'))  # Set location on earth
me.doframe(me.epoch('utc','16jun1999/12:30:20'))   # Set epoch
azel = me.measure(dir,'azel')       # Convert to azimuth/elevation
print 'az,el=', qa.angle(azel['m0']), qa.angle(azel['m1'])
#az,el= +105.16.05 -024.23.00
#
"""
```

In this example we first find the world coordinates of the centre of the image. Then we use the Measures \tool\ {\stf me} to convert the {\cf direction coordinate} field from J2000 to an azimuth and elevation at a particular location at a particular time.

---

image.decompose.html

image.decompose - Function

1.1.1 Separate a complex image into individual components

Description

This function is an image decomposition tool that performs several tasks, with the end result being that a strongly blended image is separated into components - both in the sense that it determines the parameters for each component (assuming a Gaussian model) and that it physically assigns each pixel in the image to an individual object. The products of these two operations are called the component list and the component map, respectively. The fitting process (which determines the component list) and the pixel-decomposition process (which determines the component map) are designed to work cooperatively to increase the efficiency and accuracy of both. The algorithm behind the decomposition is based on the function clfind, described in Williams et al 1994, which uses a contouring procedure whereby a closed contour designates a separate component. The program first separates the image into clearly distint 'regions' of blended emission, then contours each region to determine the areas constituting each component and passes this information on to the fitter, which determines the component list.
The contour deblending can optionally be replaced with a simpler local maximum scan, and the fitting can be replaced with a moment-based estimation method to speed up calculations on very large images or if either primary method causes trouble, but in general this will impede the accuracy of the fit.
The function works with both two and three dimensional images.
The return value is a record (or dictionary) that has 3 keys: `'components'`, `'blc'`, `'trc'`.
The `'components'` element is a matrix each row of which contains the gaussian parameters of the component fitted.
The `'blc'` element is a matrix of the bottom left corners (blc) of the regions found. Each row correspond to a region blc.
The `'trc'` element is a matrix of the top right corners (trc) of the regions found. Each row correspond to a region trc.
Please Note that the returned blc's and trc's are relative to `region` defined by the user. A `blc` of [0,0] implies the bottom left of the region selected and not the bottom left of the image. Obviously if no region is defined then it is the bottom left of the image.

Arguments

Inputs	
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: variant Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
simple	Skip contour deblending and scan for local maxima allowed: bool Default: false
threshold	Value of minimum positive contour. Must be set and nonnegative. allowed: double Default: -1
ncontour	Number of contours to use in deblending (>= 2) allowed: int Default: 11
minrange	Minimum number of closed contours in a component (> 0) allowed: int Default: 1
naxis	Max number of perpendicular steps between contiguous pixels. Values of 1, 2 or 3 are allowed. allowed: int Default: 2
fit	Fit to the components after deblending? allowed: bool Default: true
maxrms	Maximum RMS of fit residuals to not retry fit (> 0). Default is unset. allowed: double Default: -1
maxretry	Maximum number of times to retry the fit (>= 0). Default is unset. allowed: int Default: -1
maxiter	Maximum number of iterations allowed in a single fit (> 0) allowed: int Default: 256
convcriteria	Criterion to establish convergence (>=0) allowed: double Default: 0.0001
stretch	Stretch the mask if necessary and possible? See help par.stretch. 122 allowed: bool Default: false

Returns
record

Example

```
"""
#
print "\t----\t decompose Ex 1 \t----"
ia.maketestimage()
out=ia.decompose(threshold=2.5, maxrms=1.0)
#Attempt 1: Converged after 21 iterations
#Attempt 1: Converged after 15 iterations
#1: Peak: 17.955  Mu: [0.000327928, 8.62573e-05]
#               Axes: [0.00175981, 0.00142841]  Rotation: 1.29539
#2: Peak: 19.8093  Mu: [1.67927e-06, -0.000374393]
#                Axes: [0.00179054, 0.00132541]  Rotation: 1.78404
#3: Peak: 10.1155  Mu: [6.28252, -7.09688e-05]
#                Axes: [0.00180877, 0.00104523]  Rotation: 1.78847
print out['components']
#[[  1.79549522e+01   3.27928370e-04   8.62573434e-05   1.75980886e-03
#    8.11686337e-01   1.29538655e+00]
# [  1.98093319e+01   1.67927124e-06  -3.74393392e-04   1.79054437e-03
#    7.40229547e-01   1.78403902e+00]
# [  1.01155214e+01   6.28252172e+00  -7.09688029e-05   1.80877140e-03
#    5.77867746e-01   1.78847444e+00]]
print out['blc']
#[[37 31]
# [47 25]
# [67 33]]
print out['trc']
#[[54 47]
# [66 38]
# [78 40]]
ia.close()
#
"""
```

image.deconvolvecomponentlist.html

image.deconvolvecomponentlist - Function

1.1.1 Deconvolve a componentlist from the restoring beam

Description

This method deconvolves (a record representation of) a Componentlist tool from the restoring beam, returning (a record representation of) a new Componentlist tool. If there is no restoring beam, a fail is generated. Currently, only deconvolution of Gaussian components is supported.
For images with per-plane beam, the user must choose which beam is used for the deconvolution by setting channel and/or polarization. Only a single beam is used to deconvolve all components.
See also functions setrestoringbeam and restoringbeam.

Arguments

Inputs	
complist	Componentlist to deconvolve allowed: record Default:
channel	Zero-based channel number to use for beam for per plane images. Not used if the image has a single beam. allowed: int Default: -1
polarization	Zero-based polarization number to use for beam for per plane images. Not used if the image has a single beam. allowed: int Default: -1

Returns
record

Example

```
"""
#
print "\t----\t deconvolvecomponentlist Ex 1 \t----"
ia.maketestimage()
r = ia.fitcomponents()
cl1 = r['results']                     # cl1 and cl2 are record representations
r = ia.fitcomponents()
cl1 = r['results']                      # cl1 and cl2 are record representations
cl2 = ia.deconvolvecomponentlist(cl1)  #   of componentlists
print cl1, cl2
cl.fromrecord(cl2)                     # set componentlist tool with record
ia.close()
cl.close()
#
"""
```

---

image.deconvolvefrombeam.html

image.deconvolvefrombeam - Function

1.1.1 Helper function to deconvolve the given source Gaussian from a beam Gaussian to return a model Gaussian

Description

This is a helper function. It is to provide a way to deconvolve gaussians from other gaussians if that is what is needed for example removing a beam Gaussian from a Gaussian source. To run this function the tool need not be attached to an image.
The return value is a record that contains the fit param and the return value is a boolean which is set to true if fit model is a point source

Arguments

Inputs	
source	Three quantities that define the source majoraxis, minoraxis and Position angle allowed: any Default: variant
beam	Three quantities that define the beam majoraxis, minoraxis and Position angle allowed: any Default: variant

Returns
record

Example

```
"""
#
```

```
print "\t----\t deconvolvefrombeam Ex 1 \t----"
ia.maketestimage()
recout=ia.deconvolvefrombeam(source=['5arcmin', '3arcmin', '20.0deg'], beam=['50arcsec','30a
ia.close()
print 'Is pointsource ', recout['return']
print 'major=',recout['fit']['major']
print 'minor=',recout['fit']['minor']
print 'pa=',recout['fit']['pa']



"""
```

---

image.beamforconvolvedsize.html

image.beamforconvolvedsize - Function

1.1.1 Determine the size of the beam necessary to convolve with the given source to reach the given convolved (source+beam) size

Description

Determine the size of the beam necessary to convolve with the given source to reach the given convolved (source+beam) size. Because the problem is completely specified by the input parameters, no image needs to be attached to the associated tool; eg ia.open() need not be called prior to calling this method.

Arguments

Inputs	
source	Three quantities that define the deconvolved source major axis, minor axis and position angle allowed: any Default: variant
convolved	Three quantities that define the convolved source (source+beam) major axis, minor axis and position angle. Do not specify if beam is specified. allowed: any Default: variant

Returns

record

Example

```
# get the beam necessary to convolve the specified source with to achieve the target convolv
beam = ia.beamforconvolvedsize(source=["1arcsec", "1arcsec", "0deg"], convolved="3arcsec", '
```

image.commonbeam.html

image.commonbeam - Function

1.1.1 Determine a beam to which all beams in an image can be convolved.

Description

Determine a beam to which all beams in an image can be convolved. If the image does not have a beam, an exception will be thrown. If the image has a single beam, that beam will be returned. If the image has multiple beams, this will be the beam with the largest area in the image beam set if all the other beams can be convolved to that beam. If not, this is guaranteed to be the minimum area beam to which all beams in the set can be convolved if all but one of the beams in the set can be convolved to the beam in the set with the largest area. Otherwise, the returned beam may or may not be the smallest possible beam to which all the beams in the set can be convolved.

Arguments

Returns
record

Example

```
ia.open("mymultibeamimage.im")
cb = ia.commonbeam()
# convolve all the planes in the image with that beam
ia.convolve2d(outfile="myconvolvedimage.im", major=cb["major], minor=cb["minor"], pa=cb
```

image.remove.html

image.remove - Function

1.1.1 Delete the image file associated with this image tool

Description

This function first closes the `image tool` which detaches it from its underlying `image file`. It then deletes that `image file`. If `done=False`, the `image tool` is still viable, and can be used with function open to open a new `image file`. Otherwise the `image tool` is destroyed. If `verbose=True`, the logger will receive a progress report.

Arguments

Inputs		
done	Destroy this tool after deletion	
	allowed:	bool
	Default:	false
verbose	Send a progress report to the logger.	
	allowed:	bool
	Default:	true

Returns

bool

Example

```
"""
#
print "\t----\t remove Ex 1 \t----"
ia.maketestimage('myimage',overwrite=true)
ia.close()
ia.maketestimage('myotherimage',overwrite=true)
ia.close()
ia.open('myimage')          # Attach to 'myimage'
```

```
ia.remove(F)                # Close imagetool and delete ‘myimage’
ia.open(’myotherimage’)     # Open new imagefile ‘myotherimage’
ia.remove()
print "!!!EXPECT THE FOLLOWING TO GENERATE AN ERROR MESSAGE!!!"
ia.open(’myimage’)          # ’myimage’ was deleted above
ia.close()
#
"""
```

---

image.removefile.html

image.removefile - Function

1.1.1 Delete an unattached image file from disk. Note: use remove() if the image file is attached to the image tool.

Description

This function deletes the specified image file.

Arguments

Inputs	
file	Name of image file/directory to be removed. Must be specified. allowed: string Default:

Returns

bool

Example

```
"""
#
print "\t----\t removefile Ex 1 \t----"
ia.maketestimage('myimage',overwrite=true)
ia.close()
ia.removefile('myimage')                    # remove image 'myimage'
ia.maketestimage('myimage',overwrite=false) # error here if 'myimage' exists
ia.close()
ia.removefile('myimage')
#
"""
```

image.done.html

image.done - Function

1.1.1 Destroy this image tool

Description

When the user no longer needs to use an `image tool`, calling this function will free up its resources. That is, it destroys the `tool`. This means that the user can no longer call any functions on the `tool` after it has been `done`.
If the Image `tool` is associated with a disk file, then (unlike the `close` function, the user can also choose to delete that by setting `remove=true`. By default, any associated disk file is not deleted.
Note that this function is different from the `close` function because the latter does not destroy the `image tool`. For example, the user can use the `open` function straight after the `close` function on the same `tool`.

Arguments

Inputs		
remove	Delete the associated disk file as well?	
	allowed:	bool
	Default:	false
verbose	Send a progress report to the logger?	
	allowed:	bool
	Default:	true

Returns
bool

Example

```
"""
#
print "\t----\t done Ex 1 \t----"
# Make a test image and create tool subim:
```

```
ia.maketestimage('myfile',overwrite=true)
subim = ia.subimage('myfile2',overwrite=true)
# Check that subim exists as intended by attempting to display its summary:
subim.summary()      # This displays a summary of the dataset.
# Use done to destroy the subim tool:
subim.done()
# Check that the subim tool has been detached as intended, by attempting to display its summ
subim.summary()      # This should now throw an error.
ia.summary()     # This still works, though, as the ia tool is still open, and the dataset i
ia.close()
#
"""
```

---

image.fft.html

image.fft - Function

1.1.1 FFT the image

Description

This function fast Fourier Transforms the supplied image to the Fourier plane. Both float valued and complex valued images are supported. If the `axes` parameter is left unset, then the sky plane of the image (if there is one) is transformed. Otherwise, the user can specify which axes are to be transformed. Note that if a sky axis is to be transformed, both of them must be specified. The user specifies which form is desired in the result by specifying the desired output image file name(s).
Before the FFT is performed, any masked pixels are set to values of zero. The output mask is the combination (logical OR) of the default input `pixel mask` (if any) and the OTF mask. Any other input `pixel masks` will not be copied. The function maskhandler can be used if there is a need to copy other masks too.

Arguments

Inputs	
real	Output real image file name. allowed: string Default:
imag	Output imaginary image file name. allowed: string Default:
amp	Output amplitude image file name. allowed: string Default:
phase	Output phase image file name. allowed: string Default:
axes	Specify the pixel axes that are to undergo the FFT. The default option (-1) is to transform the sky plane(s). allowed: intArray Default: -1
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	The mask to be used. See "help par.mask" for more details. The default option is none. allowed: any Default: variant
stretch	Stretch the mask if it is necessary and possible. See "help par.stretch" for more details. allowed: bool Default: false
complex	Output complex valued image file name. allowed: string Default:

Returns
bool

Example

```
"""
#
print "\t----\t fft Ex 1 \t----"
# Create a test image:
ia.maketestimage('gc.small', overwrite=true)
# Perform an FFT on the sky plane of the test image,
# writing out just the resulting real and amplitude images:
ia.fft(real='r.im', amp='a.im')
# Close the image tool when done:
ia.close()
# Lastly, clean up the example output files:
ia.removefile('r.im')
ia.removefile('a.im')
#
"""
```

Example

```
"""
#
print "\t----\t fft Ex 2 \t----"
# Create a zero-filled 3D test dataset and add noise to it:
ia.fromshape('gc.small', [64,64,128], overwrite=true)
ia.addnoise(type='normal', pars=[0.5, 1], zero=false)
# The following transforms only the third axis of the image.
# writing out only the amplitude and phase images.
ia.fft(amp='amp.im', phase='p.im', axes=[2])
# Close the image tool when done:
ia.close()
# Lastly, clean up the example output files:
ia.removefile('amp.im')
ia.removefile('p.im')
#
"""
```

image.findsources.html

image.findsources - Function

1.1.1 Find point sources in the sky

Description

This function finds strong point sources in the image. The sources are returned in a record that can be used by a Componentlist `tool`.
An efficient method is used to locate sources under the assumption that they are point-like and not too close to the noise. Only sources with a peak greater than the `cutoff` fraction of the strongest source will be found. Only positive sources will be found, unless the `negfind=T` whereupon positive and negative sources will be found.
After the list of point sources has been made, you may choose to make a Gaussian fit for each one (`point=F`) so that shape information can be recovered as well. You can specify the half-width of the fitting grid with argument `width` which defaults to 5 (fitting grid would then be [11,11] pixels). If you set `width=0`, this is a signal that you would still like Gaussian components returned, but a default width should be used for the Gaussian shapes. The default is such that the component is circular with a FWHM of `width` pixels. Thus, if `point=T`, the components in the returned Componentlist are Point components. If `point=F` then Gaussian components are returned.
The `region-of-interest` must be 2-dimensional and it must hold a region of the sky. Any degenerate trailing dimensions in the region are discarded.
See also the function fitcomponents (for which `findsources` can provide an initial estimate).

Arguments

Inputs		
nmax	Maximum number of sources to find, > 0	
	allowed:	int
	Default:	20
cutoff	Fractional cutoff level	
	allowed:	double
	Default:	0.1
region	Region selection. See "help par.region" for details. Default is to use the full image.	
	allowed:	any
	Default:	variant
mask	Mask to use. See help par.mask. Default is none.	
	allowed:	any
	Default:	variant
point	Find only point sources?	
	allowed:	bool
	Default:	true
width	Half-width of fit grid when point=F	
	allowed:	int
	Default:	5
negfind	Find negative sources as well as positive?	
	allowed:	bool
	Default:	false

Returns
record

Example

```
"""
#
print "\t----\t findsources Ex 1 \t----"
ia.maketestimage()
clrec = ia.findsources(nmax=5, cutoff=0.5)
print clrec
#
"""
```

```
All sources stronger than 0.5 of the strongest will be found.
We use the Componentlist GUI to look at the strongest component.
```

---

image.fitprofile.html

image.fitprofile - Function

1.1.1 Fit gaussians and/or polynomials to a 1-dimensional profile.

Description

This application simultaneously fits any number of gaussian singlets, any number of lorentzian singlets, and any number of gaussian multiplets, and/or a polynomial to one dimensional profiles using the non-linear, least squares Levenberg-Marquardt algorithm. A description of the fitting algorithm may be found in AIPS++ Note 224 (http://www.astron.nl/casacore/trunk/casacore/doc/notes/224.html) and in Numerical Recipes by W.H. Press et al., Cambridge University Press. A gaussian/lorentzian singlet is a gaussian/lorentzian whose parameters (amplitude, center position, and width) are all independent from any other feature that may be simultaneously fit. A gaussian multiplet is a set of two or more gaussian lines in which at least one (and possibly two or three) parameter of each line is dependent on the parameter of another, single (reference) profile in the multiplet. For example, one can specify a doublet in which the amplitude of the first line is 0.6 times the amplitude of the zeroth line and/or the center of the first line is 20 pixels from the center of the zeroth line, and/or the fwhm of the first line is identical (in pixels) to that of the zeroth line. There is no limit to the number of components one can specify in a multiplet (except of course that the number of parameters to be fit should be significantly less than the number of data points), but there can be only a single reference profile in a multiplet to which to tie constraints of parameters of the other profiles in the set.
Additionally, a power logarithmic polynomial (plp) or a logarithmic tranformed polynomial (ltp) can be fit. In this case, each of these functions cannot be fit simultaneously with any other supported function. These functions are most often used for fitting the spectral index and higher order terms of a spectrum. A power logarithmic polynomial has the form
y = c0*x/div**(c1 + c2*ln(x/div) + c3*ln(x/div)**2 + ... +
cn*ln(x/div)**(n - 1))
and a logarithmic transformed polynomial is simply the result of this equation after taking the natural log of both sides so that it has the form
ln(y) = c0 + c1*ln(x/div) + c2*ln(x/div)**2 + ... + cn*ln(x/div)**n
The coefficients of the two forms correspond with each other except that c0 in the second equation is equal to ln(c0) of the first. In the case of fitting a spectral index, the spectral index, traditionally represented as alpha, is equal to c1.

In both cases, div is a numerical value used to scale abscissa values so they are closer to unity when they are sent to the fitter. This generally improves the probability that the fit will converge. This parameter may be specified via the div parameter. A value of 0 (the default) indicates that the application should determine a reasonable value for div, which is determined via
div = 10**int(log10(sqrt(min(x)*max(x))))
where min(x) and max(x) are the minimum and maximum abscissa values, respectively.
So, for example, if S(nu) is proportional to nu**alpha and you expect alpha to be near -0.8 and the value of S(nu) is 1.5 at 1e9 Hz and your image(s) have spectral units of Hz, you would specify spxest=[1.5, -0.8] and div=1e9 when fitting a plp function, or spxest=[0.405, -0.8] and div=1e9 if fitting an ltp function.
More details of fitting all of these functions are described in following sections.
A CAUTIONARY NOTE Note that the likelihood of getting a reliable solution increases with the number of good data points as well as the goodness of the initial estimate. It is possible that the first solution found might not be the best one, and so, if a solution is found, it is recommended that the fit be repeated using the solution of the previous fit as the initial estimatE for the new fit. This process should be repeated until the solutions from one fit to the next differ only insignificantly. The convergent solution is very likely the best solution.
AXIS The axis parameter indicates on which axis profiles should be fit; a value ¡0 indicates the spectral axis should be used, or if one does not exist, that the zeroth axis should be used.
MINIMUM NUMBER OF PIXELS The minpts parameter indicates the minimum number of unmasked pixels that must be present in order for a fit to be attempted. When multifit=T, positions with too few good points will be masked in any output images.
ONE FIT OF REGION AVERAGE OR PIXEL BY PIXEL FIT The multifit parameter indicates if profiles should be fit at each pixel in the selected region (true), or if the profiles in that region should be averaged and the fit done to that average profile (false).
POLYNOMIAL FITTING The order of the polynomial to fit is specified only via the poly parameter. If poly¡0, no polynomial will be fit. No initial estimates of coefficients can be specified; these are determined automatically.
GAUSSIAN SINGLET FITTING In the absence of an estimates file and no estimates being specified by the p*est parameters, and gmncomps=0 or is empty, the ngauss parameter indicates the maximum number of gaussian singlets that should be fit. The initial estimates of the parameters for these gaussians will be attempted automatically in this case. If it deems appropriate, the fitter will fit fewer than this number. In the case where an estimates file is supplied, ngauss is ignored (see below). ngauss is also ignored if the p*est parameters are specified or if gmncomps is not an empty array or, if an integer, is greater than zero. If estimates is not specified or the p*est parameters are not specified and ngauss=0, gmncomps is empty or 0, and

poly¡0, an error will occur as this indicates there is nothing to fit.
One can specify initial estimates of gaussian singlet parameters via an estimates file or the pampest, pcenterest, pfwhmest, and optionally, the pfix parameters. The latter is the recommended way to specify these estimates as support for estimates files may be deprecated in the future. No matter which option is used, an amplitude initial estimate must always be nonzero. A negative fwhm estimate will be silently changed to positve.
SPECIFYING INITIAL ESTIMATES FOR GAUSSIAN AND LORENTZIAN SINGLETS (RECOMMENDED METHOD) One may specify initial estimates via the pampest, pcenterest, and pfwhmest parameters. In the case of a single gaussian or lorentzian singlet, these parameters can be numbers. pampest must be specified in image brightness units, pcenterest must be given in the number of pixels from the zeroth pixel, and pfwhmest must be given in pixels. Optionally pfix can be specified and in the case of a single gaussian or lorentzian singlet can be a string. In it is coded which parameters should be held constant during the fix. Any combination of "p" (amplitude), "c" (center), or "f" (fwhm) is allowed; eg pfix="pc" means fix both the amplitude and center during the fit. In the case of more than one gaussian and/or lorentzian singlets, these parameters must be specified as arrays of numbers. The length of the arrays indicates the number of singlets to fit and must be the same for all the p*est parameters.
If no parameters are to be fixed for any of the singlets, pfix can be set to the empty string. However, if at least one parameter of one singlet is to be fixed, pfix must be an array of strings and have a length equal to the p*est arrays. Singlets which are not to have any parameters fixed should be represented as an empty string in the pfix array. So, for example, if one desires to fit three singlets and fix the fwhm of the middle one, one must specify pfix=["", "f", ""], the empty strings indicating no parameters of the zeroth and second singlet should be held constant.
In the case of multifit=True, the initial estimates, whether from the p*est parameters or from a file (see below), will be applied to the location of the first fit. This is normally the bottom left corner of the region selected. If masked, not enough good points to perform a fit, or the attempted fit fails, the fitting proceeds to the next pixel with the pixel value of the lowest numbered axis changing the fastest. Once a successful fit has been performed, subsequent fits will use the results of a fit for a nearest pixel for which a previous fit was successful as the initial estimate for the parameters at the current location. The fixed parameter string will be honored for every fit performed when multifit=True.
One specifies what type of PCF profile to fit via the pfunc parameter. A PCF function is one that can be parameterized by a peak, center, and FWHM, as both gaussian and lorentzian singlets can. If all singlets to be fit are gaussians, one can set pfunc equal to the empty string and all snglets will be assumed to be gaussians. If at least one lorentzian is to be fit, pfunc must be specified as a string (in the case of a single singlet) or an array of strings (in the case of multiple singlets). The position of each string corresponds to the positions of

the initial estimates in the p*est and pfix arrays. Minimal match ("g", "G", "l", or "L") is supported. So, if one wanted to simultaneously fit two gaussian and two lorentzian singlets, the zeroth and last of which were lorentzians, one would specify pfunc=["L", "G", "G", "L"].
ESTIMATES FILE FOR GAUSSIAN SINGLETS (NONRECOMMENDED METHOD) Initial estimates for gaussian singlets can be specified in an estimates file. Estimates files may be deprecated in the future in favor of the p*est parameters, so it is recommended users use those parameters instead. If an estimates file is desired to be used, the p*est parameters must be 0 or empty and mgncomps must be 0 or empty. Only gaussian singlets can be specified in an estimates file. If one desires to fit one or more gaussian multiplets and/or one or more lorentzian singlets simultaneously, the p*est parameters must be used to specify the initial parameters of all gaussian singlets to fit; one cannot use an estimates file in this case. If an estimates file is specified, a polynomial can be fit simultaneously by specifying the poly parameter. The estimates file must contain initial estimates of parameters for all gaussian singlets to be fit. The number of gaussian singlets to fit is gotten from the number of estimates in the file. The file can contain comments which are indicated by a "#" at the beginning of a line. All non-comment lines will be interpreted as initial estimates. The format of such a line is
[peak intensity], [center], [fwhm], [optional fixed parameter string]
The first three values are required and must be numerical values. The peak intensity must be expressed in image brightness units, while the center must be specified in pixels offset from the zeroth pixel, and fwhm must be specified in pixels. The fourth value is optional and if present, represents the parameter(s) that should be held constant during the fit. Any combination of the characters 'p' (peak), 'c' (center), and 'f' (fwhm) are permitted, eg "fc" means hold the fwhm and the center constant during the fit. Fixed parameters will have no error associated with them. Here is an example file:

```
# estimates file indicating that two gaussians should be fit
# first guassian estimate, peak=40, center at pixel number 10.5, fwhm = 5.8 pixels, all para
# fit
40, 10.5, 5.8
# second gaussian, peak = 4, center at pixel number 90.2, fwhm = 7.2 pixels, hold fwhm const
4, 90.2, 7.2, f
# end file
```

GAUSSIAN MULTIPLET FITTING Any number of gaussian multiplets, each containing any number of two or more components, can be simultaneously fit, optionally with a polynomial and/or any number of gaussian and/or lorentzian singlets, the only caveat being that the number of parameters to be fit should be significantly less than the number of data points. The gmncomps parameter indicates the number of multiplets to fit and the number of components in each multiplet. In the case of a single multiplet, an integer (¿1) can be specified. For example, mgncomps=4 means fit a single quadruplet of

gaussians. In the case of 2 or more multiplets, and array of integers (all ¿1) must be specified. For example, gmncomps=[2, 4, 3] means 3 seperate multiples are to be fit, the zeroth being a doublet, the first being a quadruplet, and the second being a triplet.
Initial estimates of all gaussians in all multiplets are specified via the gm*est parameters which must be arrays of numbers. The order starts with the zeroth component of the zeroth multiplet to the last component of the zeroth multiplet, then the zeroth component of the first multiplet to the last compoenent of the first multiplet, etc to the zeroth component of the last multiplet to the last element of the last multiplet. The zeroth element of a multiplet is defined as the reference component of that multiplet and has the special significance that it is the profile to which all constraints of all other profiles in that multiplet are referenced (see below). So, in our example of gmncomps=[2, 4, 3], gmampest, gmcenterest, and gmfwhmest must each be nine (the total number of individual gaussian profiles summed over all multiplets) element arrays. The zeroth, second, and sixth elements represent parameters of the reference profiles in the zeroth, first, and second multiplet, respectively.
The fixed relationships between the non-reference profile(s) and the reference profile of a multiplet are specified via the gmampcon, gmcentercon, and gmfwhmcon parameters. At least one, and any combination, of constraints can be specified for any non-reference component of a multiplet. The amplitude ratio of a non-reference line to that of the reference line is set in gmampcon. The ratio of the fwhm of a non-reference line to that of the reference line is set in gmfwhmcon. The offset in pixels of the center position of a non-reference line to that of the reference line is set in gmcentercon. In the case where a parameter is not constrained for any non-reference line of any multiplet, the value of the associated parameter must be 0. In the case of a single doublet, a constraint may be specified as a number or an array of a single number. For example, mgncomps=2 and gmampcon=0.65 and gmcentercon=[32.4] means there is a single doublet to fit where the amplitude ratio of the first to the zeroth line is constained to be 0.65 and the center of the first line is constrained to be offset by 32.4 pixels from the center of the zeroth line. In cases of a total of three or more gaussians, the constraints parameters must be specified as arrays with lengths equal to the total number of gaussians summed over all multiplets minus the number of reference lines (one per multiplet, or just number of multiplets, since reference lines cannot be constrained by themselves). In the cases where an array must be specified but a component in that array does not have that constraint, 0 should be specified. Here's an example
gmncomps=[2, 4, 3] gmampcon= [0 , 0.2, 0 , 0.1, 4.5, 0] gcentercon=[24.2, 45.6, 92.7, 0 , -22.8, -33.5] gfwhmcon=""
In this case we have our previous example of one doublet, one quadruplet, and one triplet. The first component of the doublet has the constraint that its center is offset by 24.2 pixels from the zeroth (reference) component. The first component of the quadruplet is constrained to have an amplitude of 0.2 times

that of the quadruplet's zeroth component and its center is constrained to be offset by 45.6 pixels from the reference component. The second component of the quadruplet is constained to have its center offset by 92.7 pixels from the associated reference component and the third component is constrained to have an amplitude of 0.1 times that of the associated reference component. The first component of the triplet is constrained to have an amplitude of 4.5 times that of its associated reference component and its center is constrained to be offset by -22.8 pixels from the reference component's center. The second component of the triplet is constrained to have its center offset by -33.5 pixels from the center of the reference component. No lines have FWHM constraints, so the empty string can be given for that parameter. Note that using 0 to indicate no constraint for line center means that one cannot specify a line centered at the same position as the reference component but having a different FWHM from the reference component. If you must specify this very unusual case, try using a very small positive (or even negative) value for the center constraint.

Note that when a parameter for a line is constrained, the corresponding value for that component in the corresponding gm*est array is ignored and the value of the constrained parameter is automatically used instead. So let's say, for our example above, we had specified the following estimates:

gmampest = [1, .2, 2, .1, .1, .5, 3, 2, 5] gmcenterest = [20, 10 , 30, 45.2, 609 , -233, 30, -859, 1]

Before any fitting is done, the constraints would be taken into account and these arrays would be implicitly rewritten as:

gmampest = [1, .2, 2, .4, .1, .2, 3, 13.5, 5] gmcenterest = [20, 44.2, 30, 75.6, 127.7, -233, 30, 7.2, -3.5]

The value of gmfwhmest would be unchanged since there are no FWHM constraints in this example.

In addition to be constrained by values of the reference component, parameters of individual components can be fixed. Fixed parameters are specified via the gmfix parameter. If no parameters are to be fixed, gmfix can be specified as the empty string or a zero element array. In the case where any parameter is to be fixed, gmfix must be specified as an array of strings with length equal to the total number of components summed over all multiplets. These strings encode which parameters to be fixed for the corresponding components. If a component is to have no parameters fixed, an empty string is used. In other cases one or more of any combination of parameters can be fixed using "p", "c", and/or "f" described above for fixing singlet parameters. There are a couople of special cases to be aware of. In the case where a non-reference component parameter is constrained and the corresponding reference component parameter is set as fixed, that parameter in the non-reference parameter will automatically be fixed even if it was specified not to be fixed in the gmfix array. This is the only way the constraint can be honored afterall. In the converse case of when a constrained parameter of a non-reference component is specified as fixed, but the corresponding parameter in the reference component is not specified to be fixed, an error will occur.

Fixing an unconstrained parameter in a non-reference component is always legal as is fixing any combination of parameters in a reference component (with the above caveat that corresponding constrained parameters in non-reference components will be silently held fixed as well).
The same rules that apply to singlets when multifit=True apply to multiplets.
LIMITING RANGES FOR SOLUTION PARAMETERS In cases of low (or no) signal to noise spectra, it is still possible for the fit to converge, but often to a nonsensical solution. The astronomer can use her knowledge of the source to filter out obviously bogus solutions. Any solution which contains a NaN value as a value or error in any one of its parameters is automatically marked as invalid.
One can also limit the ranges of solution parameters to known "good" values via the goodamprange, goodcenterrange, and goodfwhmrange parameters. Any combination can be specified and the limit constraints will be ANDed together. The ranges apply to all PCF components that might be fit; choosing ranges on a component by component basis is not supported. If specified, an array of exactly two numerical values must be given to indicate the range of acceptable solution values for that parameter. goodamprange is expressed in terms of image brightness units. goodcenterrange is expressed in terms of pixels from the zeroth pixel in the specified region. goodfwhmrange is expressed in terms of pixels (only non-negative values should be given for FWHM range endpoints). In the case of a multiple-PCF fit, if any of the corresponding solutions are outside the specified ranges, the entire solution is considered to be invalid.
In addition, solutions for which the absolute value of the ratio of the amplitude error to the amplitude exceeds 100 or the ratio of the FWHM error to the FWHM exceeds 100 are automatically marked as invalid.
POWER LOGARITHMIC POLYNOMIAL AND LOGARITHMIC TRANSFORMED POLYNOMIAL FITTING Fitting of a sngle power logarithmic polynomial or a single logarithmic transformed polynomial function is supported. No other functions may be fit simultaneously with either of these; if parameters relating to other functions are supplied simultaneously with parameters relating to these functions, an exception will occur. For details of the functional forms, see the introduction of this document.
The set of c0 ... cn coefficients (as defined previously) can be solved for. Initial estimates for the c values should be supplied via the plpest or ltpest parameters, depending on which form is being fit. The number of values given in this array will be the number of coeffecients that are solved for. One may specify which coefficients should be held fixed during the fit in the plpfix or ltpfix array. If supplied, this array should have the same number of elements as its respective initial estimates array. A value of True means the corresponding coefficient will be held fixed during the fit. An empty array indicates that no parameters will be held fixed. This is the default.
Because the logarithm of the ordinate values must be taken before fitting a logarithmic transformed polynomial, all non-positive pixel values are effectively masked for the purposes of fitting.

INCLUDING STANDARD DEVIATIONS OF PIXEL VALUES If the standard deviations of the pixel values in the input image are known and they vary in the image (eg they are higher for pixels near the edge of the band), they can be included in the sigma parameter. This parameter takes either an array or an image name. The array or image must have one of three shapes: 1. the shape of the input image, 2. the same dimensions as the input image with the lengths of all axes being one except for the fit axis which must have length corresponding to its length in the input image, or 3. be one dimensional with lenght equal the the length of the fit axis in the input image. In cases 2 and 3, the array or pixels in sigma will be replicated such that the image that is ultimately used is the same shape as the input image. The values of sigma must be non-negative. It is only the relative values that are important. A value of 0 means that pixel should not be used in the fit. Other than that, if pixel A has a higher standard deviation than pixel B, then pixel A is noisier than pixel B and will receive a lower weight when the fit is done. The weight of a pixel is the usual
weight = 1/(sigma*sigma)
In the case of multifit=F, the sigma values at each pixel along the fit axis in the hyperplane perpendicular to the fit axis which includes that pixel are averaged and the resultant averaged standard deviation spectrum is the one used in the fit. Internally, sigma values are normalized such that the maximum value is 1. This mitigates a known overflow issue.
One can write the normalized standard deviation image used in the fit but specifying its name in outsigma. This image can then be used as sigma for subsequent runs.
RETURNED DICTIONARY STRUCTURE The returned dictionary has a (necessarily) complex structure. First, there are keys "xUnit" and "yUnit" whose values are the abscissa unit and the ordinate unit described by simple strings. Next there are arrays giving a broad overview of the fit quality. These arrays have the shape of the specified region collapsed along the fit axis with the axis corresponding to the fit axis having length of 1:
attempted: a boolean array indicating which fits were attempted (eg if too few unmasked points, a fit will not be attempted). converged: a boolean array indicating which fits converged. False if the fit was not attempted. valid: a boolean array indicating which solutions fall within the specified valid ranges of parameter space (see section LIMITING RANGES FOR SOLUTION PARAMETERS for details). niter: an int array indicating the number of iterations for each profile, ¡0 if the fit did not converge ncomps: the number of components (gaussian singlets + lorentzian singlets + gaussian multiplets + polynomial) fit for the profile, ¡0 if the fit did not converge direction: a string array containing the world direction coordinate for each profile
There is a "type" array having number of dimensions equal to the number of dimensions in the above arrays plus one. The shape of the first n-1 dimensions is the same as the shape of the above arrays. The length of the last dimension is equal to the number of components fit. The values of this array are strings describing the components that were fit at each possition ("POLYNOMIAL",

"GAUSSIAN" in the case of gaussian singlets, "LORENTZIAN" in the case of lorentzian singlets, and ""GAUSSIAN MULTPLET").
If any gaussian singlets were fit, there will be a subdictionary accessible via the "gs" key which will have subkeys "amp", "ampErr", "center", "centerErr", "fwhm", "fwhmErr, "integral", and "integralErr". Each of these arrays will have one more dimension than the overview arrays described above. The shape of the first n-1 dimensions will be the same as the shape of the arrays described above, while the final dimension will have length equal to the maximum number of gaussian singlets that were fit. Along this axis will be the corresponding fit result or associated error (depending on the array's associated key) of the fit for that singlet component number. In cases where the fit did not converge, or that particular component was excluded from the fit, a value of NAN will be present.
If any lorentzian singlets were fit, their solutions will be accessible via the "ls" key. These arrays follow the same rules as the "gs" arrays described above.
If any gaussian multiplets were fit, there will be subdictionaries accessible by keys "gm0", "gm1", ..., "gmn-1" where n is the number of gaussian muliplets that were fit. Each of these dictionaries will have the same arrays described above for gaussian singlets. The last dimension will have length equal to the number of components in that particular multiplet. Each pixel along the last axis will be the parameter solution value or error for that component number in the multiplet, eg the zeroth pixel along that axis contains the parameter solution or error for the reference component of the multiplet.
The polynomial coefficient solutions and errors are not returned, although they are logged.
If a power logarithmic polynomial was fit, there will be a subdictionary accessible via the "plp" key which will have subkeys "soltuion" and "error" which will each have an array value. Each of these arrays will have one more dimension than the overview arrays described above. The shape of the first n-1 dimensions will be the same as the shape of the overview arrays described above, while the final dimension will have length equal to the number of parameters that were fit. Along this axis will be the corresponding fit result or associated error (depending on the array's associated key) of the fit. In cases where the fit was not attempted or did not converge, a value of NAN will be present.
OUTPUT IMAGES In addition to the returned dictionary, optionally one or more of any combination of output images can be written. The model and residual parameters indicate the names of the model and residual images to be written; blank values inidcate that these images should not be written.
One can also write none, any or all of the solution and error images for gaussian singlet, lorentzian singlet, and gaussian multiplet fits via the parameters amp, amperr, center, centererr, fwhm, fwhmerr, integral, integralerr when doing multi-pixel fits. For a power logarithmic polynomial or a logarithmic transformed polynomial fit, plpsol or ltpsol and plperr or ltpsol are the names of the solution and error images to write, respectively.
These images contain the arrays described for the associated parameter

solutions or errors described in previous sections. Each component is written to a different image, and each image is distiguished by the component it represents by its name ending in an uderscore and the relevant component number ("_0", "_1", etc). In the case of Gaussian multiplets, the image name ends with the number of the mulitplet group followed by the number of the component in that group (eg "_3_4" represents component 4 in multiplet group 3). In the case of lorentzian singlets, "_ls" is appended to the image names (but before the identifying component number), in the case of gaussian multiplets. Similarly "_gm" is included in the name of Gaussian multiplet images. Pixels for which fits were not attempted, did not converge, or converged but have values of NaN (not a number) or INF (infinity) will be masked as bad. Writing analogous images for polynomial coefficients is not supported.

Arguments

Inputs	
box	Rectangular region to select in direction plane. See "help par.box" for details. Default is to use the entire direction plane. allowed: string Default:
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
chans	Channels to use. See "help par.chans" for details. Channels must be contiguous. Default is to use all channels.. allowed: string Default:
stokes	Stokes planes to use. See "help par.stokes" for details. Planes must be contiguous. Default is to use all stokes planes. allowed: string Default:
axis	The profile axis. Default: use the spectral axis if one exists, axis 0 otherwise (<0). allowed: int Default: -1
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
ngauss	Number of Gaussian elements. Default: 1. allowed: int Default: 1
poly	Order of polynomial element. Default: do not fit a polynomial (<0). allowed: int Default: -1
estimates	Name of file containing initial estimates. Default: No initial estimates (""). allowed: string Default:
minpts	Minimum number of unmasked points necessary to attempt fit. allowed: int Default: 1
multifit	If true, fit a profile along the desired axis at each pixel in the specified region. If false, average the non-fit axis pixels and do a single fit to that average profile. Default False. allowed: bool Default: false
model	Name of model image. Default: do not write the model image (""). allowed: string Default:
residual	Name of residual image. Default: do not write the residual image (""). allowed: string

Returns
record

Example

```
"""
ia.open("myspectrum.im")
res = ia.fitprofile(ngauss=2, box="3,3,4,5", poly=2, multifit=true)
"""
```

---

image.fitcomponents.html

image.fitcomponents - Function

1.1.1 Fit 2-dimensional models to an image.

Description

OVERVIEW
This application is used to fit one or more two dimensional gaussians to sources in an image as well as an optional zero-level offset. Fitting is limited to a single polarization but can be performed over several contiguous spectral channels. If the image has a clean beam, the report and returned dictionary will contain both the convolved and the deconvolved fit results.
When dooff is False, the method returns a dictionary with three keys, 'converged', 'results', and 'deconvolved'. The value of 'converged' is a boolean array which indicates if the fit converged on a channel by channel basis. The value of 'results' is a dictionary representing a component list reflecting the fit results. In the case of an image containing beam information, the sizes and position angles in the 'results' dictionary are those of the source(s) convolved with the restoring beam, while the same parameters in the 'deconvolved' dictionary represent the source sizes deconvolved from the beam. In the case where the image does not contain a beam, 'deconvolved' will be absent. Both the 'results' and 'deconvolved' dictionaries can be read into a component list tool (default tool is named cl) using the fromrecord() method for easier inspection using tool methods, eg
cl.fromrecord(res['results'])
although this currently only works if the flux density units are conformant with Jy.
There are also values in each component subdictionary not used by cl.fromrecord() but meant to supply additional information. There is a 'peak' subdictionary for each component that provides the peak intensity of the component. It is present for both 'results' and 'deconvolved' components. There is also a 'sum' subdictionary for each component indicated the simple sum of pixel values in the the original image enclosed by the fitted ellipse. There is a 'channel' entry in the 'spectrum' subdictionary which provides the zero-based channel number in the input image for which the solution applies. In addtion, if the image has a beam(s), then there will be a 'beam' subdictionary associated with each component in both the 'results' and 'deconvolved' dictionaries. This subdictionary will have three keys: 'beamarcsec' will be a subdictionary giving the beam dimensions in arcsec, 'beampixels' will have the value of the beam area expressed in pixels, and 'beamster' will have the value of the beam area epressed in steradians. Also, if

the image has a beam(s), in the component level dictionaries will be an 'ispoint' entry with an associated boolean value describing if the component is consistent with a point source.
If dooff is True, in addtion to the specified number of gaussians, a zero-level offset will also be fit. The initial estimate for this offset is specified using the offset parameter. Units are assumed to be the same as the image brightness units. The zero level offset can be held constant during the fit by specifying fixoffset=True. In the case of dooff=True, the returned dictionary contains two additional keys, 'zerooff' and 'zeroofferr', which are both dictionaries containing 'unit' and 'value' keys. The values associated with the 'value' keys are arrays containing the the fitted zero level offset value and its error, respectively, for each channel. In cases where the fit did not converge, these values are set to NaN. The value associated with 'unit' is just the image brightness unit.
The region can either be specified by a box(es) or a region. Ranges of pixel values can be included or excluded from the fit. If specified using the box parameter, multiple boxes can be given using the format box="blcx1, blcy1, trcx1, trcy1, blcx2, blcy2, trcx2, trcy2, ... , blcxN, blcyN, trcxN, trcyN" where N is the number of boxes. In this case, the union of the specified boxes will be used.
If specified, the residual and/or model images for successful fits will be written.
If an estimates file is not specified, an attempt is made to estimate initial parameters and fit a single Gaussian. If a multiple Gaussian fit is desired, the user must specify initial estimates via a text file (see below for details).
The user has the option of writing the result of the fit to a log file, and has the option of either appending to or overwriting an existing file.
The user has the option of writing the (convolved) parameters of a successful fit to a file which can be fed back to fitcomponents() as the estimates file for a subsequent run.
If specified and positive, the value of rms is used to calculate the parameter uncertainties, otherwise, the rms in the selected region in the relevant channel is used for these calculations.
The noisefwhm parameter represents the noise-correlation beam FWHM. If specified as a quantity, it should have angular units. If specified as a numerical value, it is set equal to that number of pixels. If specified and greater than or equal to the pixel size, it is used to calculate parameter uncertainties using the correlated noise equations (see below). If it is specified but less than a pixel width, the the uncorrelated noise equations (see below) are used to compute the parameter uncertainties. If it is not specified and the image has a restoring beam(s), the the correlated noise equations are used to compute parameter uncertainties using the geometric mean of the relevant beam major and minor axes as the noise-correlation beam FWHM. If noisefwhm is not specified and the image does not have a restoring beam, then the uncorrelated noise equations are used to compute the parameter uncertainties.
SUPPORTED UNITS
Currently only images with brightness units conformant with Jy/beam,

Jy.km/s/beam, and K are fully supported for fitting. If your image has some other base brightness unit, that unit will be assumed to be equivalent to Jy/pixel and results will be calculated accordingly. In particular, the flux density (reported as Integrated Flux in the logger and associated with the "flux" key in the returned component subdictionary(ies)) for such a case represents the sum of pixel values.
Note also that converting the returned results subdictionary to a component list via cl.fromrecord() currently only works properly if the flux density units in the results dictionary are conformant with Jy. If you need to be able to run cl.fromrecord() on the resulting dictionary you can first modify the flux density units by hand to be (some prefix)Jy and then run cl.fromrecord() on that dictionary, bearing in mind your unit conversion.
If the input image has units of K, the flux density of components will be reported in units of [prefix]K*rad*rad, where prefix is an SI prefix used so that the numerical value is between 1 and 1000. To convert to units of K*beam, determine the area of the appropriate beam, which is given by pi/(4*ln(2))*bmaj*bmin, where bmaj and bmin are the major and minor axes of the beam, and convert to steradians (=rad*rad). This value is included in the beam portion of the component subdictionary (key 'beamster'). Then divide the numerical value of the logged flux density by the beam area in steradians. So, for example

```
# run on an image with K brightness units
res = imfit(...)
# get the I flux density in K*beam of component 0
comp = res['results']['component0']
flux_density_kbeam = comp['flux']['value'][0]/comp['beam']['beamster']
```

FITTING OVER MULTIPLE CHANNELS
For fitting over multiple channels, the result of the previous successful fit is used as the estimate for the next channel. The number of gaussians fit cannot be varied on a channel by channel basis. Thus the variation of source structure should be reasonably smooth in frequency to produce reliable fit results.
MASK SPECIFICATION
Mask specification can be done using an LEL expression. For example mask = '"myimage"¿5' will use only pixels with values greater than 5.
INCLUDING AND EXCLUDING PIXELS
Pixels can be included or excluded from the fit based on their values using these parameters. Note that specifying both is not permitted and will cause an error. If specified, both take an array of two numeric values.
ESTIMATES
Initial estimates of fit parameters may be specified via an estimates text file. Each line of this file should contain a set of parameters for a single gaussian. Optionally, some of these parameters can be fixed during the fit. The format of each line is
peak intensity, peak x-pixel value, peak y-pixel value, major axis, minor axis, position angle, fixed

The fixed parameter is optional. The peak intensity is assumed to be in the same units as the image pixel values (eg Jy/beam). The peak coordinates are specified in pixel coordinates. The major and minor axes and the position angle are the convolved parameters if the image has been convolved with a clean beam and are specified as quantities. The fixed parameter is optional and is a string. It may contain any combination of the following characters 'f' (peak intensity), 'x' (peak x position), 'y' (peak y position), 'a' (major axis), 'b' (minor axis), 'p' (position angle).
In addition, lines in the file starting with a # are considered comments.
An example of such a file is:

```
# peak intensity must be in map units
120, 150, 110, 23.5arcsec, 18.9arcsec, 120deg
90, 60, 200, 46arcsec, 23arcsec, 140deg, fxp
```

This is a file which specifies that two gaussians are to be simultaneously fit, and for the second gaussian the specified peak intensity, x position, and position angle are to be held fixed during the fit.
ERROR ESTIMATES
Error estimates are based on the work of Condon 1997, PASP, 109, 166. Key assumptions made are: * The given model (elliptical Gaussian, or elliptical Gaussian plus constant offset) is an adequate representation of the data * An accurate estimate of the pixel noise is provided or can be derived (see above). For the case of correlated noise (e.g., a CLEAN map), the fit region should contain many "beams" or an independent value of rms should be provided. * The signal-to-noise ratio (SNR) or the Gaussian component is large. This is necessary because a Taylor series is used to linearize the problem. Condon (1997) states that the fractional bias in the fitted amplitude due to this assumption is of order 1/(S*S), where S is the overall SNR of the Gaussian with respect to the given data set (defined more precisely below). For a 5 sigma "detection" of the Gaussian, this is a 4% effect. * All (or practically all) of the flux in the component being fit falls within the selected region. If a constant offset term is simultaneously fit and not fixed, the region of interest should be even larger. The derivations of the expressions summarized in this note assume an effectively infinite region.
Two sets of equations are used to calculate the parameter uncertainties, based on if the noise is correlated or uncorrelated. The rules governing which set of equations are used have been described above in the description of the noisefwhm parameter.
In the case of uncorrelated noise, the equations used are
f(A) = f(I) = f(M) = f(m) = k*s(x)/M = k*s(y)/m = (s(p)/sqrt(2))*((M*M - m*m)/(M*m)) = sqrt(2)/S
where s(z) is the uncertainty associated with parameter z, f(z) = s(z)/abs(z) is the fractional uncertainty associated with parameter z, A is the peak intensity, I is the flux density, M and m are the FWHM major and minor axes, p is the position angle of the component, and k = sqrt(8*ln(2)). s(x) and s(y) are the

direction uncertainties of the component measured along the major and minor axes; the resulting uncertainties measured along the principle axes of the image direction coordinate are calculated by propagation of errors using the 2D rotation matrix which enacts the rotation through the position angle plus 90 degrees. S is the overall signal to noise ratio of the component, which, for the uncorrelated noise case is given by
S = (A/(k*h*r))*sqrt(pi*M*m)
where h is the pixel width of the direction coordinate and r is the rms noise (see the discussion above for the rules governing how the value of r is determined).
For the correlated noise case, the same equations are used to determine the uncertainties as in the uncorrelated noise case, except for the uncertainty in I (see below). However, S is given by
S = (A/(2*r*N)) * sqrt(M*m) * (1 + ((N*N/(M*M)))**(a/2)) * (1 + ((N*N/(m*m)))**(b/2))
where N is the noise-correlation beam FWHM (see discussion of the noisefwhm parameter for rules governing how this value is determined). "**" indicates exponentiation and a and b depend on which uncertainty is being calculated. For sigma(A), a = b = 3/2. For M and x, a = 5/2 and b = 1/2. For m, y, and p, a = 1/2 and b = 5/2. f(I) is calculated in the correlated noise case according to
f(I) = sqrt(f(A)*f(A) + (N*N/(M*m))*(f(M*f(M) + f(m)*f(m))))
Note well the following caveats: * Fixing Gaussian component parameters will tend to cause the parameter uncertainties reported for free parameters to be overestimated. * Fitting a zero level offset that is not fixed will tend to cause the reported parameter uncertainties to be slightly underestimated. * The parameter uncertainties will be inaccurate at low SNR (a ∼10% for SNR = 3). * If the fitted region is not considerably larger than the largest component that is fit, parameter uncertainties may be mis-estimated. * An accurate rms noise measurement, r, for the region in question must be supplied. Alternatively, a sufficiently large signal-free region must be present in the selected region (at least about 25 noise beams in area) to auto-derive such an estimate. * If the image noise is not statistically independent from pixel to pixel, a reasonably accurate noise correlation scale, N, must be provided. If the noise correlation function is not approximately Gaussian, the correlation length can be estimated using
N = sqrt(2*ln(2)/pi)* double-integral(dx dy C(x,y))/sqrt(double-integral(dx dy C(x, y) * C(x,y)))
where C(x,y) is the associated noise-smoothing function * If fitted model components have significan spatial overlap, the parameter uncertainties are likely to be mis-estimated (i.e., correlations between the parameters of separate components are not accounted for). * If the image being analyzed is an interferometric image with poor uv sampling, the parameter uncertainties may be significantly underestimated.
The deconvolved size and position angle errors are computed by taking the maximum of the absolute values of the differences of the best fit deconvolved

value of the given parameter and the deconvolved size of the eight possible combinations of (FWHM major axis +/- major axis error), (FWHM minor axis +/- minor axis error), and (position andle +/- position angle error). If the source cannot be deconvolved from the beam (if the best fit convolved source size cannot be deconvolved from the beam), upper limits on the deconvolved source size are sometimes reported. These limits simply come from the maximum major and minor axes of the deconvolved gaussians taken from trying all eight of the aforementioned combinations. In the case none of these combinations produces a deconvolved size, no upper limit is reported.
EXAMPLE:
Here is how one might fit two gaussians to multiple channels of a cube using the fit from the previous channel as the initial estimate for the next. It also illustrates how one can specify a region in the associated continuum image as the region to use as the fit for the channel.

```
imagename = "co_cube.im"
# specify region using region from continuum
region = "continuum.im:source.rgn"
chans = "2~20"
# only use pixels with positive values in the fit
excludepix = [-1e10,0]
# estimates file contains initial parameters for two Gaussians in channel 2
estimates = "initial_estimates.txt"
logfile = "co_fit.log"
# append results to the log file for all the channels
append = "True"
ia.open(imagename)
ia.fitcomponents(region=region, chans=chans, excludepix=excludepix, estimates=estimates, log
```

Arguments

Inputs	
box	Rectangular region(s) to select in direction plane. See "help par.box" for details. Default is to use the entire direction plane. allowed: string Default:
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
chans	Channels to use. See "help par.chans" for details. Default is 0 (first plane). allowed: any Default: variant
stokes	The stokes planes to use. See "help par.stokes" for details. Default is to use the first stokes plane. allowed: string Default:
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
includepix	Range of pixel values to include. Default is to include all pixels. allowed: doubleArray Default: -1
excludepix	Range of pixel values to exclude. Default is to exclude no pixels. allowed: doubleArray Default: -1
residual	Name of the residual image to write. Default is not to write the residual. allowed: string Default:
model	Name of the model image to write. Default is not to write the model. allowed: string Default:
estimates	Name of the input estimates file. Default is to autoestimate in which case a single gaussian will be fit. allowed: string Default:
logfile	File in which to log results. Default is not to write a logfile. allowed: string Default:
append	Append results to logfile? Logfile must be specified. Default is to append. False means overwrite existing file if it exists. allowed: bool Default: true
newestimates	File to which to write results in "estimates" format suitable as estimates input for another run. Default is do not write an estimates file.

Returns
record

---

image.fromrecord.html

image.fromrecord - Function

1.1.1 Generate an image from a record

Description

You can convert an associated image to a record (torecord) or imagepol tool functions will sometimes give you a record. This function (fromrecord) allows you to set the contents of an image tool to the content of the record

Arguments

Inputs		
record	Record containing the image	
	allowed:	record
	Default:	
outfile	The name of the diskfile to be created for image from record	
	allowed:	string
	Default:	

Returns

bool

Example

```
"""
#
print "\t----\t fromrecord Ex 1 \t----"
ia.maketestimage('image.large', overwrite=true)
rec=ia.torecord()
ia.close()
ia.fromrecord(rec, "testimage")
```

"""

---

image.getchunk.html

image.getchunk - Function

1.1.1 Get the pixel values from a regular region of the image into an array

Description

This function returns the pixels (or optionally the pixel mask) from the `image file` between `blc` and `trc` inclusively. Both float and complex valued images are supported. An increment may be specified with `inc`. Note that if you retrieve too many pixels, you might cause swapping since the pixels are kept in memory.
Any illegal `blc` values are set to zero. Any illegal `trc` values are set to the end of the image. If any `trc` $<$ `blc`, you get the whole image for that axis. Any illegal `inc` values are set to unity.
The argument `axes` can be used to reduce the dimensionality of the output array. It specifies which pixel axes of the image to **average** the data over. For example, consider a 3-D image. With `axes=[0,1]` and all other arguments left at their defaults, the result would be a 1-D vector, a profile along the third axis, with the data averaged over the first two axes.
A related function is getregion which retrieves the pixels or `pixel mask` from a potentially more complex `region-of-interest`. Function `getchunk` is retained because it is faster and therefore preferable for repeated operation in loops if the `pixel mask` is not required and the region is a simple box.
If you set getmask=T, the return value is the 'pixelmask' rather than the 'pixel' image.

Arguments

Inputs		
blc	Bottom-Left-Corner (beginning) of pixel section. Default is start of image.	
	allowed:	intArray
	Default:	-1
trc	Top-Right-Corner (end) of pixel section. Default is end of image.	
	allowed:	intArray
	Default:	-1
inc	increment (stride) along axes	
	allowed:	intArray
	Default:	1
axes	Axes to average over. Default is none.	
	allowed:	intArray
	Default:	-1
list	List bounding box to logger?	
	allowed:	bool
	Default:	false
dropdeg	Drop degenerate axes?	
	allowed:	bool
	Default:	false
getmask	Get the pixel mask rather than the pixel values	
	allowed:	bool
	Default:	false

Returns
anyvariant

Example

```
Suppose that we have a 3-dimensional image called {\sff im}. Then:

"""
#
print "\t----\t getchunk Ex 1 \t----"
ia.fromshape(shape=[64,64,128])
pix = ia.getchunk()                        # all pixels
ia.calcmask('T')                           # give image a mask
pix = ia.getchunk([1,1,1], [10,10,1])      # 10 by 10 section of plane # 1
pix = ia.getchunk([1,1], [1,1])            # first spectrum
pix = ia.getchunk(inc=[1,5])               # all planes, decimated by 5 in y
```

```
mask = ia.getchunk(getmask=T)            # Get pixelmask
ia.close()
#
"""
```

---

image.getregion.html

image.getregion - Function

1.1.1 Get pixels or mask from a region-of-interest of the image

Description

This function recovers the image pixel or `pixel mask` values in the given region-of-interest. Regardless of the shape of the `region` you have specified, the shape of the `pixels` and `pixelmask` arrays must necessarily be the bounding box of the specified region. If the region extends beyond the image, it is truncated.
Recall that the recovered `pixel mask` will reflect both the `pixel mask` stored in the image, and the `region-of-interest` (their masks are 'anded') – see the discussion in the introduction about this.
The argument `axes` can be used to reduce the dimensionality of the output array. It specifies which pixel axes of the image to average the data over. For example, consider a 3-D image. With `axes=[0,1]` and all other arguments left at their defaults, the result would be a 1-D vector, a profile along the third axis, with the data averaged over the first two axes.
This function differs in three ways from `getchunk`. First, the region can be much more complex (e.g. a union of polygons) than the simple `blc`, `trc`, and `inc` of `getchunk` (although such a region can be created of course). Second, it can be used to recover the `pixel mask` or the pixels. Third, it is less efficient than `getchunk` for doing the same thing as `getchunk`. So if you are interested in say, iterating through an image, getting a regular hyper-cube of pixels and doing something with them, then `getchunk` will be faster. This would be especially noticeable if you iterated line by line through a large image.

Arguments

Inputs	
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
axes	Axes to average over. Default is none. allowed: intArray Default: -1
mask	Mask to use. See help par.mask. Default is none. allowed: variant Default:
list	List the bounding box to the logger allowed: bool Default: false
dropdeg	Drop degenerate axes allowed: bool Default: false
getmask	Get the pixel mask rather than pixel values allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false

Returns
variant

Example

```
Suppose that we have a 3-dimensional image called {\sff cube} and wish
to recover the pixel from a simple regular region.

"""
#
print "\t----\t getregion Ex 1 \t----"
ia.fromshape('cube', [64,64,64], overwrite=true)
#r1=rg.box(blc=[10,10,10],trc=[30,40]) # Create region
r1=rg.box([10,10,10],[30,40,40]) # Create region
pixels=ia.getregion(r1)
```

```
ia.close()
#
"""
```

Example

```
"""
#
print "\t----\t getregion Ex 2 \t----"
ia.fromshape('cube', [64,64,64], overwrite=true)
pixels = ia.getregion()
pixelmask = ia.getregion(getmask=T)
#
"""
```

In this example we recover first the pixels and then the pixel mask.

---

image.getprofile.html

image.getprofile - Function

1.1.1 Get values and mask for a one dimensional profile along a specified image axis by applying an aggregate function.

Description

This application returns information on a one-dimensional profile taken along a specified image axis. The region of interest is collapsed (a'la ia.collapse() along all axes orthogonal to the one specified, and) the specified aggregate function is applied to these pixels to generate the returned values.
The aggregate function must be one of the functions supported by ia.collapse; ie, 'flux', 'max', 'mean', 'median', 'min', 'rms', 'stdev', 'sum', and 'variance'. See the help for ia.collapse() for details regarding these functions. Minimum match and case insenstivity is supported.
One may specify the unit of the returned coordinate values. Unless axis is the spectral axis, unit must be conformant with the corresponding axis unit in the image coordinate system or it must be 'pixel' which signifies, pixel, rather than world, coordinate values should be calculated. If axis is the spectral axis, unit may be a velocity unit (assuming the coordinate system has a rest frequency or restfreq is specified) or a length unit. In these cases, the returned coordinate values will be converted to velocity or wavelength, respectively.
The parameter spectype may be used to specify the velocity or wavelength type for the returned coordinate values if profile is taken along spectral axis. Supported (minimum match, case insensitive) values) are "relativistic velocity", "beta", "radio velocity", "optical velocity", "wavelength", "air wavelength", "default". The "default" value is equivalent to "relativistic" if unit is a velocity unit or "wavelength" if unit is a length unit.
The restfreq parameter allows one to set the rest frequency for the coordinates to be returned if axis is the spectral axis and unit is a velocity unit. If blank, the rest frequency associated with the image coordinate system is used.
The frame allows one to specify which kinematic reference frame that the returned coordinate values should be calculated in. It is only used if axis is the spectral axis and unit is unspecified or is specified and a frequency unit. If blank, the reference frame associated with the image coordinate system is used.
The returned dictionary contains the keys:
values: one-dimensional array along the specified axis containing values resulting from applying the specified aggregate function to corresponding pixels at the same location along that axis. mask: one-dimensional array of booleans of the resulting mask after applying the aggregate function, formed in the same way as that formed by ia.collapse. coords One-dimensional array

of corresponding coordinate values along the specified axis in the specified unit (or the unit associated with the axis in the image coordinate system if unspecified). xUnit The unit used for calculating the values the coords array.

Arguments

Inputs	
axis	Axis along which to determine profile. Must be specified allowed: int Default: -1
function	Aggregate function to apply for collapse along axes orthogonal to specified axis. allowed: string Default: mean
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: string Default:
unit	Unit of the returned abscissa values. Must either be 'pixel' or be conformant with image axis unit unless axis is the spectral axis. Default is the unit associated with axis in the image coordinate system. allowed: string Default:
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false
spectype	Velocity or wavelength type if profile taken along spectral axis. Supported (minimum match, case insensitive) values are "relativistic velocity", "beta", "radio velocity", "optical velocity", "wavelength", "air wavelength", "default". allowed: string Default: default
restfreq	Rest frequency to use when calculating coordinate values. Used only if axis is spectral axis and unit is not the unit associated with the axis in the coordinate system. Empty string means use the rest frequency associated with the image coordinate system allowed: any Default: variant
frame	Reference frame to use when calculating coordinate values. Used only if axis is spectral axis and unit is not the unit associated with the axis in the coordinate system. Empty string means use the reference frame associated with the image coordinate system allowed: string Default:
logfile	File to which to write profile. allowed: string Default:

Returns
record

Example

```
ia.open('myimage')
# get the max pixel values along axis 2
res = ia.getprofile(axis=2, function='max')

# axis 2 is the spectral axis. Get the minimum pixel values along this axis
# and specify that the returned coordinate values should be optical velocities
# in km/s

res2 = ia.getprofile(axis=2, function='min', unit='km/s', spectype='optical')

ia.done()
```

---

image.getslice.html

image.getslice - Function

1.1.1 Get 1-D slice from the image

Description

This function returns a 1-D slice (the pixels and opionally the pixel mask) from the `image file`. The slice is constrained to lie in a plane of two cardinal axes (e.g. XY or YZ). At some point this constraint will be relaxed. A range of interpolation schemes are available.
You specify the slice as a polyline giving the x (`x`) and y (`y`) coordinates and the axes of the plane holding that slice (`axes`). As well, you must specify the absolute pixel coordinates of the other axes (`coord`). This defaults to the first pixel (e.g. first plane).
The return value is a record with fields 'pixels' (interpolated intensity), 'mask' (interpolated mask), 'xpos' (x-location in absolute pixel coordinates), 'ypos' (y-location in absolute pixel coordinates), 'distance' (distance along slice in pixels), 'axes' (the x and y axes of slice).

Arguments

Inputs		
x	Polyline x vertices in absolute pixel coordinates	
	allowed:	doubleArray
	Default:	
y	Polyline y vertices in absolute pixel coordinates	
	allowed:	doubleArray
	Default:	
axes	Pixel axes of plane holding slice. Default is first two axes.	
	allowed:	intArray
	Default:	01
coord	Specify pixel coordinate for other axes. Default is first pixel.	
	allowed:	intArray
	Default:	-1
npts	Number of points in slice. Default is auto determination.	
	allowed:	int
	Default:	0
method	The interpolation method, String from 'nearest', 'linear', 'cubic'	
	allowed:	string
	Default:	linear

Returns
record

Example

```
Suppose that we have a 2-dimensional image. Then:

"""
#
print "\t----\t getslice Ex 1 \t----"
ia.maketestimage();
rec = ia.getslice (x=[1,20], y=[2,30])     # SLice from [1,2] -> [20,30]
print rec.keys()
#['distance', 'xpos', 'axes', 'mask', 'ypos', 'pixel']
rec = ia.getslice (x=[1,20,25,11], y=[2,30,32,40]) # Polyline slice
ia.close()
#
"""
```

image.hanning.html

image.hanning - Function

1.1.1 Convolve one axis of image with a Hanning kernel

Description

This application performs Hanning convolution of one axis of an image defined by
z[i] = 0.25*y[i-1] + 0.5*y[i] + 0.25*y[i+1] (equation 1)
where z[i] is the value at pixel i in the hanning smoothed image, and y[i-1], y[i], and y[i+1] are the values of the input image at pixels i-1, i, and i+1 respectively. It supports both float and complex valued images. The length of the axis along which the convolution is to occur must be at least three pixels in the selected region. Masked pixel values are set to zero prior to convolution. All nondefault pixel masks are ignored during the calculation.
The convolution is done in the image domain (i.e., not with an FFT).
If drop=False, the length of the output axis will be the same as that of the input axis. The output pixel values along the convolution axis will be related to those of the input values according to equation 1, except the first and last pixels. In that case,
z[0] = 0.5*(y[0] + y[1])
and,
z[N-1] = 0.5*(y[N-2] + y[N-1])
where N is the number of pixels along the convolution aixs. The pixel mask, ORed with the OTF mask if specified, is copied from the selected region of the input image to the output image. Thus for example, if the selected region in the input image has six planes along the convolution axis, and if the pixel values, which are all unmasked, on a slice along this axis are [1, 2, 5, 10, 17, 26], the corresponding output pixel values will be [1.5, 2.5, 5.5, 10.5, 17.5, 21.5].
If drop=True and dmethod="copy", the output image is the image calculated if drop=True, except that only the odd-numbered planes are kept.
Furthermore, if the number of planes along the convolution axis in the selected region of the input image is even, the last odd number plane is also discarded. Thus, if the selected region has N pixels along the convolution axis in the input image, along the convolution axis the output image will have (N-1)/2 planes if N is odd, or (N-2)/2 planes if N is even. In this case, the pixel and mask values are copied directly, without further processing. Thus for example, if the selected region in the input image has six planes along the convolution axis, and if the pixel values, which are all unmasked, on a slice along this axis are [1, 2, 5, 10, 17, 26], the corresponding output pixel values will be [2.5, 10.5].

If drop=True and dmethod="mean", first the image described in the drop=False case is calculated. The first plane and last plane(s) of that image are then discarded as described in the drop=True, dmethod="copy" case. Then, the ith plane of the output image is calculated by averaging the (2*i)th and (2*i + 1)th planes of the intermediate image.Thus for example, if the selected region in the input image has six planes along the convolution axis, and if the pixel values, which are all unmasked, on a slice along this axis are [1, 2, 5, 10, 17, 26], the corresponding output pixel values will be [4.0, 14.0]. Masked values are taken into consideration when forming this average, so if one of the values is masked, it is not used in the average. If at least one of the values in the input pair is not masked, the corresponding output pixel will not be masked.
The hanning smoothed image is written to disk with name `outfile`, if specified. If not, no image is written but the image is still accessible via the returned image analysis tool (see below).
This method always returns an image analysis tool which is attached to the hanning smoothed image. This tool should always be captured and closed after any desired manipulations have been done. Closing the tool frees up system resources (eg memory), eg,
hanning_image = ia.hanning(...)

```
# do things (or not) with hanning_image
...
# close the returned tool promptly upon finishing with it.
```

hanning_image.done()
See also the other convolution functions convolve2d, sepconvolve and convolve.

Arguments

Inputs	
outfile	Output image file name. Default is unset. allowed: string Default:
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
axis	Zero based axis to convolve. ia.coordsys().names() gives the order of the axes in the image. Less than 0 means use the spectral axis if there is one, if not an exception is thrown. allowed: int Default: -10
drop	Drop every other pixel on output? allowed: bool Default: true
overwrite	Overwrite (unprompted) pre-existing output file? allowed: bool Default: false
async	Run asynchronously? allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false
dmethod	If drop=True, method to use in plane decimation. "c(opy)": direct copy of every second plane, "m(ean)": average planes 2*i and 2*i+1 in the smoothed, non-decimated image to form plane i in the output image. allowed: string Default: copy

Returns
image

Example

```
ia.open("mynonsmoothed.im")
# smooth the spectral axis, say it's axis 2 and only write every other pixel
hanning = ia.hanning(outfile="myhanningsmoothed.im", axis=2, drop=True, overwrite=True)
# done with input
ia.done()
# do something with the output image, get statistics say
stats = hanning.statistics()
# close the result image
hanning.done()
```

---

image.haslock.html

image.haslock - Function

1.1.1 Does this image have any locks set?

Description

This function can be used to find out whether the image has a read or a write lock set. It is not of general user interest. It returns a vector of Booleans of length 2. Position 1 says whether a read lock is set, position 2 says whether a write lock is set.
In general locking is handled automatically, with a built in lock release cycle. However, this function can be useful in scripts when a file is being shared between more than one process. See also functions unlock and lock.

Arguments

Returns
boolArray

Example

```
"""
#
print "\t----\t haslock Ex 1 \t----"
ia.maketestimage('xx',overwrite=true)
ia.lock(writelock=T)
print ia.haslock()
#[True, True]
ia.unlock()
print ia.haslock()
#[False, False]
ia.lock(F)
```

```
print ia.haslock()
#[True, False]
ia.close()
#
"""

This example acquires a read/write lock on the file and then unlocks it
and acquires just a read lock.
```

---

image.histograms.html

image.histograms - Function

1.1.1 Compute histograms from the image

Description

This method computes histograms of the pixel values in the image. The values are returned in a dictionary.
The chunk of the image over which you compute the histograms is specified by a vector of axis numbers (argument `axes`). For example, consider a 3-dimensional image for which you specify `axes=[0,2]`. The histograms would be computed for each XZ (axes 0 and 2) plane in the image. You could then examine those histograms as a function of the Y (axis 1) axis. Or perhaps you set `axes=[2]`, whereupon you could examine the histogram for each Z (axis 2) profile as a function of X and Y location in the image.
You have control over the number of bins for each histogram (`nbins`). The bin width is worked out automatically for each histogram and may vary from histogram to histogram (the range of pixel values is worked out for each chunk being histogrammed).
You have control over which pixels are included in the histograms via the `includepix` argument. This vector specifies a range of pixel values to be included in the histograms. If you only give one value for this, say `includepix=[b]`, then this is interpreted as `includepix=[-abs(b),abs(b)]`. If you specify an inclusion range, then the range of pixel intensities over which the histograms are binned is given by this range too. This is a way to make the bin width the same for each histogram.
You can control if the histogram is cumulative or non-cumulative via the cumu parameter.
You have countrol over how the bin counts are returned. If log = false, the actual counts are returned. If true, the values returned are the log10 values of the actual counts.
The results are returned as a dictionary. The counts (field "counts") and the abscissa values (field "values") for all bins in each histogram are returned. The shape of the first dimension of those arrays contained in those fields is `nbins`. The number and shape of the remaining dimensions are those of the display axes(the axes in the image for which you did not compute the histograms). For example, if one has a three dimensional image and sets `axes=[2]`, the display axes are 0 and 1, so the shape of each counts and values array is then [nbins,nx,ny], where nx and ny are the length of the zeroth and first axes, respectively.

In addition, the mean (field "mean") and standard deviation (field "sigma") computed using the data in each histogram is returned. The shape of these arrays is equal to the shape of the display axes. So,

Arguments

Inputs	
axes	List of axes to compute histograms over. Default is all axes. allowed: intArray Default: -1
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
nbins	Number of bins in histograms, > 0 allowed: int Default: 25
includepix	Range of pixel values to include. Default is to include all pixels. allowed: doubleArray Default: -1
cumu	If T the bin values are cumulative. allowed: bool Default: false
log	If true, the returned counts values will be the log10 values of the actual counts, if false, the actual counts will be returned. allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false

Returns
record

Example

```
# obtain a histogram using the entire image
ia.maketestimage()
res = ia.histograms()
ia.close()

# obtain histograms for each plane along axis 1 with each
# histogram having 30 bins. Only pixel values in the range
# -0.001 to 0.001 are used in computing the histograms and the
# statistics. The counts in the returned data structure represent
# the cumulative number of data points in the current bin and in
# bins less than the current bin.
ia.open("myimage.im")
r = ia.histograms(axes=[0,2],nbins=30,includepix=1e-3,cumu=T)
ia.close()
```

---

image.history.html

image.history - Function

1.1.1 Recover and/or list the history file

Description

This function allows you to access the history file.
If `browse=F` and `list=F`, the history is returned by the function as a vector of strings. If `list=T`, the history is sent to the logger.
CASA tools that modify the MeasurementSet or an image file will save history information. Also, you can directly annotate the history file with the function sethistory. History from FITS file conversions is also stored and listable here.

Arguments

Inputs		
list	List history to the logger?	
	allowed:	bool
	Default:	true

Returns
stringArray

Example

```
"""
#
print "\t----\t history Ex 1 \t----"
ia.maketestimage()
ia.history()                        # List history to logger
h = ia.history(list=F)              # Recover history in variable h
ia.history(list=T, browse=F)        # List history to logger
#
"""
```

image.insert.html

image.insert - Function

1.1.1 Insert specified image into this image

Description

This function inserts the specified image (or part of it) into the image referenced by this tool. The specified image may be given via argument `infile` as a disk file name (it may be in native CASA, FITS, or Miriad format; Look here for more information on foreign images).
If the `locate` vector is not given, then the images are aligned (to an integer pixel shift) by their reference pixels.
If `locate` vector is given, then those values that are given, give the absolute pixel in the output (this) image of the bottom left corner of the input (sub)image. For those values that are not given, the input image is symmetrically placed in the output image.
The image referenced by this tool is modified in place; no new image is created. The method returns True if successful.

Arguments

Inputs		
infile	Name of image to be inserted.	
	allowed:	string
	Default:	
region	Region selection. See "help par.region" for details. Default is to use the full image.	
	allowed:	any
	Default:	variant
locate	Location of input image in output image. Default is centrally located.	
	allowed:	doubleArray
	Default:	-1
verbose	Emit informational messages to logger?	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
"""
#
print "\t----\t insert Ex 1 \t----"
ia.maketestimage('myfile.insert',overwrite=true)
ia.close()
ia.fromshape(shape=[200,200])
ia.insert(infile='myfile.insert')       # Align by reference pixel
ia.newimagefromfile('myfile.insert')
ia.insert(infile=im2.name(), locate=[]) # Align centrally
# This time align axis 0 as given and axis 1 centrally
ia.insert(infile='myfile.insert', locate=[20])
ia.close()                                    # close default tool and
"""
```

---

image.isopen.html

image.isopen - Function

1.1.1 Is this Image `tool` open?

Description

This function can be used to find out whether the Image `tool` is associated with an image or not.

Arguments

Returns
bool

Example

```
"""
#
print "\t----\t isopen Ex 1 \t----"
ia.maketestimage('zz',overwrite=true)
print ia.isopen()
#True
ia.close()
print ia.isopen()
#False
ia.open('zz')
print ia.isopen()
#True
ia.close()
#
"""
```

image.ispersistent.html

image.ispersistent - Function

1.1.1 Is the image persistent?

Description

This function can be used to find out whether the image is persistent on disk or not. There is a subtle difference from the image being virtual. For example, a virtual image which references another which is on disk is termed persistent.

Arguments

Returns
bool

Example

```
"""
#
print "\t----\t ispersistent Ex 1 \t----"
ia.fromshape(outfile='tmp', shape=[10,20], overwrite=true)
print ia.ispersistent()
#True
ia.close()
ia.fromimage(infile='tmp')
print ia.ispersistent()
#True
im3 = ia.subimage()
print im3.ispersistent()            # Persistent virtual image !
#True
im4 = ia.imagecalc(pixels='tmp+tmp')
print im4.ispersistent()
```

```
#False
im3.done()
im4.done()
ia.close(remove=true)
#
"""
```

---

image.lock.html

image.lock - Function

1.1.1 Acquire a lock on the image

Description

This function can be used to acquire a Read or a Read/Write lock on the `image file`. It is not of general user interest.
In general locking is handled automatically, with a built in lock release cycle. However, this function can be useful in scripts when a file is being shared between more than one process. See also functions unlock and haslock.

Arguments

Inputs		
writelock	Acquire a read/write (T) or a readonly (F) lock	
	allowed:	bool
	Default:	false
nattempts	Number of attempts, > 0. Default is unlimiited.	
	allowed:	int
	Default:	0

Returns
bool

Example

```
"""
#
print "\t----\t lock Ex 1 \t----"
ia.maketestimage('xx', overwrite=true)
ia.lock(writelock=T)
ia.unlock()
ia.lock(writelock=F)
ia.close(remove=true)
```

```
#
"""

This acquires a read/write lock on the file. Then we unlock it
and acquire a readonly lock.
```

---

image.makecomplex.html

image.makecomplex - Function

1.1.1 Make a complex image

Description

This function combines the current image with another image to make a complex image. The current image (i.e. that associated with this Image `tool` is assumed to be the Real image). You supply the Imaginary image; it must be disk-based at this time.
The output image cannot be associated with an Image `tool` (does not handle Complex images yet) and so the best you can do is write it to disk. The Viewer can view it.

Arguments

Inputs		
outfile	Output Complex (disk) image file name	
	allowed:	string
	Default:	
imag	Imaginary image file name	
	allowed:	string
	Default:	
region	Region selection. See "help par.region" for details. Default is to use the full image.	
	allowed:	any
	Default:	variant
overwrite	Overwrite (unprompted) pre-existing output file?	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
"""
#
print "\t----\t makecomplex Ex 1 \t----"
ia.maketestimage('imag.im',overwrite=true)  #imaginary image
ia.close()
ia.maketestimage('real.im',overwrite=true)  #assoc. real image
ia.makecomplex('complex.im', 'imag.im', overwrite=true)
ia.close()
#
"""
```

---

image.maskhandler.html

image.maskhandler - Function

1.1.1 Handle pixel masks

Description

This function is used to manage or handle `pixel masks` . A CASA image may contain zero, one or more `pixel masks`. Any of these masks can be designated the default `pixel mask`. The default mask is acted upon by CASA applications. For example, if you ask for statistics from an image, pixels which are masked as bad (F) will be excluded from the calculations.
This function has an argument (`op`) that specifies the behaviour. In all cases, you can shorten the operation string to three characters. It is not the job of this function to modify the values of masks.

- default - this retrieves the name of the default `pixel mask` as the return value of the function call.
- get - this retrieves the name(s) of the existing `pixel masks` as the return value of the function call (string or vector of strings).
- set - this lets you change the default `pixel mask` to that given by the `name` argument. If `name` is empty, then the default mask is unset (i.e. an all good mask is effectively applied).
- delete - this lets you delete the `pixel masks` specified by the `name` argument. To delete more than one mask, `name` can be a vector of strings. Any supplied `pixel mask` name that does not exist is silently ignored.
- rename - this lets you rename the mask specified by `name[0]` to `name[1]`. Thus the `name` argument must be a vector of length 2.
- copy - this lets you copy a mask to another in the same image, or copy a mask from another image into this image. Thus the `name` argument must be a vector of length 2.

  For the first case, the first element of `name` must be the name of the mask to copy, and the second element must be the name of the `pixel mask` to which it will be copied.

  For the second case, the first element of `name` must be the name of the input image and `pixel mask` with a colon delimiter (e.g. `hcn:mask2`). The second element must be the name of the `pixel mask` to which the input `pixel mask` will be copied.

Use the summary function to see the available `pixel masks`. You can do this either via the logger display, or via the returned record, which contains the mask names. In the logger display, any `pixel mask` which is not the default mask is listed in square brackets. If a default mask is set, it is listed first, and is not enclosed in square brackets.

Arguments

Inputs		
op	The operation. One of 'set', 'delete', 'rename', 'get', 'copy' or 'default'	
	allowed:	string
	Default:	default
name	Name of mask or masks.	
	allowed:	stringArray
	Default:	

Returns
stringArray

Example

```
"""
#
print "\t----\t maskhandler Ex 1 \t----"
ia.maketestimage('g1.app', overwrite=true)
ia.calcmask('T', name='mask1')
ia.close()
ia.maketestimage('myimage', overwrite=true)
ia.calcmask('T')                          # Create some masks
ia.calcmask('T', name='mask1')
ia.calcmask('T', name='mask2')
names = ia.maskhandler('get')             # Get the mask names
print names
#['mask0', 'mask1', 'mask2']
name = ia.maskhandler('default')          # Get the default mask name
print name
#mask2
ia.maskhandler('set', ['mask1'])          # Make 'mask1' the default mask
```

```
ia.maskhandler('set', [''])              # Unset the default mask
ia.maskhandler('delete', ['mask1'])      # Delete 'mask1'
ia.calcmask('T', name='mask1')           # Make another 'mask1'
ia.maskhandler('delete', ['mask0', 'mask1'])# Delete 'mask0' and 'mask1'
ia.calcmask('T', name='mask1')
ia.maskhandler('rename', ['mask1', 'mask0'])# Rename 'mask1' to 'mask0'

# Copy 'mask1' from image 'g1.app'  to 'mask10' in image 'myimage'
ia.maskhandler('copy', ['g1.app:mask1', 'mask10'])
ia.removefile('g1.app')                  # Cleanup
ia.close()
#
"""
```

---

image.miscinfo.html

image.miscinfo - Function

1.1.1 Get the miscellaneous information record from an image

Description

A CASA `image file` can accumulate miscellaneous information during its lifetime. This information is stored in a record called the `miscinfo` record. For example, the FITS filler puts header keywords it doesn't otherwise use into the `miscinfo` record. This `miscinfo` record is not guaranteed to have any entries, so it's up to you to check for any fields that you require.
You can also put things into this record (see setmiscinfo) yourself, to keep information that the system might not otherwise store for you.
When the image is written out to FITS, the items in the `miscinfo` record are written to the FITS file as keywords with the corresponding record field name.

Arguments

Returns
record

Example

```
"""
#
print "\t----\t miscinfo Ex 1 \t----"
ia.maketestimage()
print ia.miscinfo()       # print the record
ia.setmiscinfo("testing")
print ia.miscinfo()
header = ia.miscinfo()    # capture the record for further use
print header
```

```
ia.close()
#
"""
```

---

image.modify.html

image.modify - Function

1.1.1 Modify image with a model

Description

This function applies a model of the sky to the image. You can add or subtract the model which is contained in a Componentlist tool.
The pixel values are only changed where the total mask (combination of the default `pixel mask` [if any] and the OTF mask) is good (True). If the computation fails for a particular pixel (e.g. coordinate undefined) that pixel will be masked bad.

Arguments

Inputs	
model	Record representation of a ComponentList model allowed: record Default:
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
subtract	Subtract or add the model allowed: bool Default: true
list	List informative messages to the logger allowed: bool Default: true
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false

Returns

bool

Example

```
"""
#
print "\t----\t modify Ex 1 \t----"
ia.maketestimage()
clrec = ia.fitcomponents()
ia.modify(clrec['results'])
ia.close()
#
"""
```

In this example we subtract the model returned by the fitcomponents function.

---

image.maxfit.html

image.maxfit - Function

1.1.1 Find maximum and do parabolic fit in the sky

Description

This function finds the pixel with the maximum value in the region, and then uses function findsources to generate a Componentlist with one component. The component will be of type Point (`point=T`) or Gaussian (`point=F`).
If `negfind=F` the maximum pixel value is found in the region and fit. If `negfind=T` the absolute maximum pixel value is found in the region and fit.
See function findsources for a description of arguments `point` and `width`.
See also the function fitcomponents.

Arguments

Inputs		
region	Region selection. See "help par.region" for details. Default is to use the full image.	
	allowed:	any
	Default:	variant
point	Find only point sources?	
	allowed:	bool
	Default:	true
width	Half-width of fit grid when point=F	
	allowed:	int
	Default:	5
negfind	Find negative sources as well as positive?	
	allowed:	bool
	Default:	false
list	List the fitted parameters to the logger?	
	allowed:	bool
	Default:	true

Returns

record

Example

```
"""
#
print "\t----\t maxfit Ex 1 \t----"
ia.maketestimage()
clrec = ia.maxfit()
print clrec          # There is only one component
ia.close()
#
"""
```

---

image.moments.html

image.moments - Function

1.1.1 Compute moments from an image

Description

Summary
The primary goal of this function is to enable you to analyze a multi-dimensional image by generating moments of a specified axis. This is a time-honoured spectral-line analysis technique used for extracting information about spectral lines.
You can generate one or more output moment images. The return value of this function is an on-the-fly Image `tool` holding the **first** of the output moment images.
The word 'moment' is used loosely here. It refers to collapsing an axis (the moment axis) to one pixel and setting the value of that pixel (for all of the other non-collapsed axes) to something computed from the data values along the moment axis. For example, take an RA-DEC-Velocity cube, collapse the velocity axis by computing the mean intensity at each RA-DEC pixel. This function offers many different moments and a variety of automatic methods to compute them.
We try to make a distinction between a 'moment' and a 'method'. This boundary is a little blurred, but it claims to refer to the distinction between what you are computing, and how the pixels that were included in that computation were selected. For example, a 'moment' would be the average value of some pixel values in a spectrum. A 'method' for selecting those pixels would be a simple pixel value range specifying which pixels should be included.
There are many available moments, and you specify each one with an integer code as it would get rather cumbersome to refer to them via strings. In the list below, the value of the ith pixel of the spectrum is I_i, the coordinate of this pixel is v_i (of course it may not be velocity), and there are n pixels in the spectrum. The available moments are:

- -1 – the mean value of the spectrum

$$\frac{1}{n}\sum I_i$$

- 0 – the integrated value of the spectrum

$$M_0 = \Delta v \sum I_i$$

where Δv is the width (in world coordinate units) of a pixel along the moment axis

- 1 – the intensity weighted coordinate (this is traditionally used to get 'velocity fields')

$$M_1 = \frac{\sum I_i v_i}{M_0}$$

- 2 – the intensity weighted dispersion of the coordinate (this is traditionally used to get 'velocity dispersion fields')

$$\sqrt{\frac{\sum I_i \left(v_i - M_1\right)^2}{M_0}}$$

- 3 – the median of I

- 4 – the median coordinate. Here we treat the spectrum as a probability distribution, generate the cumulative distribution, and then find the coordinate corresponding to the 50% value. This moment is not very robust, but it is useful for quickly generating a velocity field in a way that is not sensitive to noise. However, it will only give sensible results under certain conditions. The generation of the cumulative distribution and the finding of the 50% level really only makes sense if the cumulative distribution is monotonic. This essentially means only selecting pixels which are positive or negative. For this reason, this moment type is only supported with the basic method (see below – i.e. no smoothing, no windowing, no fitting) with a pixel selection range that is either all positive, or all negative

- 5 – the standard deviation about the mean of the spectrum

$$\sqrt{\frac{1}{(n-1)} \sum \left(I_i - \bar{I}\right)^2}$$

- 6 – the root mean square of the spectrum

$$\sqrt{\frac{1}{n} \sum I_i^2}$$

- 7 – the absolute mean deviation of the spectrum

$$\frac{1}{n}\sum|(I_i - \bar{I})|$$

- 8 – the maximum value of the spectrum
- 9 – the coordinate of the maximum value of the spectrum
- 10 – the minimum value of the spectrum
- 11 – the coordinate of the minimum value of the spectrum

Smoothing
The purpose of the smoothing functionality is purely to provide a mask. Thus, you can smooth the input image, apply a pixel include or exclude range, and generate a smoothed mask which is then applied before the moments are generated. The smoothed data are not used to compute the actual moments; that is always done from the original data.

Basic Method
The basic method is to just compute moments directly from the pixel values. This can be modified by applying pixel value inclusion or exclusion ranges (arguments `includepix` and `excludepix`).
You can then also convolve the image (arguments `smoothaxes`, `smoothtypes`, and `smoothwidths`) and find a mask based on the inclusion or exclusion ranges applied to the convolved image. This mask is then applied to the unsmoothed data for moment computation.

Window Method
The window method (invoked with argument `method='window'`) does no pixel-value-based selection. Instead a window is found (hopefully surrounding the spectral line feature) and only the pixels in that window are used for computation. This window can be found from the convolved or unconvolved image (arguments `smoothaxes`, `smoothtypes`, and `smoothwidths`).
The moments are always computed from the unconvolved data. The window can be found (for each spectrum) automatically. The automatic methods are via Bosma's converging mean algorithm (`method='window'`) or by fitting Gaussians and taking $\pm 3\sigma$ as the window (`method='window,fit'`).
In Bosma's algorithm, an initial guess for a range of pixels surrounding a spectral feature is refined by widening until the mean of the pixels outside of the range converges (to the noise).

Fit Method

The fit method (`method='fit'`) fits Gaussians to spectral features automatically. The moments are then computed from the Gaussian fits (not the data themselves).

Other Arguments

- `outfile` - If you are creating just one moment image, and you specify `outfile`, then the image is created on disk with this name. If you leave `outfile` empty then a temporary image is created. In both cases, you can access this image with the returned Image `tool`. If you are making more than one moment image, then theses images are always created on disk. If you specify `outfile` then this is the root for the output file names. If you don't specify it, then the input image name is used as the root.

- `smoothing` - If you smooth the image to generate a mask, you specify the kernel widths via the `smoothwidths` argument in the same way as in the sepconvolve function. See it for details.

- `stddev` - Some of the automatic methods also require an estimate of the noise level in the image. This is used to assess whether a spectrum is purely noise or not, and whether there is any signal worth digging out. If you don't give it via the `stddev` argument, it will be worked out automatically from a Gaussian fit to the bins above 25% from a histogram of the entire image.

- `includepix, excludepix` - The vectors given by arguments `includepix` and `excludepix` specify a range of pixel values for which pixels are either included or excluded. They are mutually exclusive; you can specify one or the other, but not both. If you only give one value for either of these, say `includepix=b`, then this is interpreted as `includepix=[-abs(b),abs(b)]`.

  The convolving point-spread function is normalized to have a volume of unity. This means that point sources are depressed in value, but extended sources that are large with respect to the PSF remain essentially on the same intensity scale; these are the structures you are trying to find with the convolution so this is what you want. If you convolve the image, then arguments like `includepix` select based upon the convolved image pixel values. If you are having trouble getting these right, you can output the convolved image (`smoothout`) and assess the validity of your pixel ranges. Note also that if you are Hanning convolving (usually used on a velocity axis), then the width for this kernel must be 3 pixels (triangular smoothing kernels of other widths have no valid theoretical basis).

- `doppler` - If you compute the moments along a spectral axis, it is conventional to compute the world coordinate (needed for moments 0, 1

and 2) along that axis in "km/s". The argument `doppler` lets you specify what doppler convention the velocity will be calculated in. You can choose from `doppler=radio, optical, true`. See function summary for the definitions of these codes. For other moment-axis types, the world coordinate is computed in the native units.

- `mask` - The total input mask is the combination of the default `pixel mask` (if any) and the OTF mask. Once this mask has been established, then the moment method may make additional pixel selections.
- `drop` - If this is true (the default) then the moment axis is dropped from the output image. Otherwise, the output images have a moment axis of unit length and coordinate information that is the same as for the input image. This coordinate information may be totally meaningless for the moment images.

Finally, if you ask for a moment which requires the coordinate to be computed for each profile pixel (these are the intensity weighted mean coordinate [moment 1] and the intensity weighted dispersion of the coordinate [moment 2]), and the profile axis is not separable then there will be a performance loss. Examples of non-separable axes are RA and Dec. If the axis is separable (e.g. a spectral axis) there is no penalty. In the latter case, the vector of coordinates for one profile is the same as the vector for another profile, and it can be precomputed (once).
Note that this function has no "virtual" output file capability. All output files are written to disk. The output mask for these images is good (T) unless the moment method fails to generate a value (e.g. the total input pixel mask was all bad for the profile) in which case it will be bad (F).
If an image has multiple (per-channel beams) and the moment axis is equal to the spectral axis, each channel will be convolved with a beam that is equal to the beam having the largest area in the beamset prior to moment determination.

Arguments

Inputs	
moments	List of moments that you would like to compute. Default is integrated spectrum. allowed: intArray Default: 0
axis	The moment axis. Default is the spectral axis if there is one. allowed: int Default: -10
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
method	List of windowing and/or fitting functions you would like to invoke. Vector of strings from 'window', 'fit' and 'interactive'. The default is to not invoke the window or fit functions, and to not invoke any interactive functions. allowed: stringArray Default:
smoothaxes	List of axes to smooth. Default is no smoothing. allowed: intArray Default: -1
smoothtypes	List of smoothing kernel types, one for each axis to smooth. Vector of strings from 'gauss', 'boxcar', 'hanning'. Default is no smoothing. allowed: any Default: variant
smoothwidths	List of widths (full width for boxcar, full width at half maximum for gaussian, 3 for Hanning) in pixels for the smoothing kernels. Vector of numeric. Default is no smoothing. allowed: doubleArray Default: 0.0
includepix	Range of pixel values to include. Vector of 1 or 2 doubles. Default is include all pixel. allowed: doubleArray Default: -1
excludepix	Range of pixel values to exclude. Default is exclude no pixels. allowed: doubleArray Default: -1
peaksnr	The SNR ratio below which the spectrum will be rejected as noise (used by the window and fit functions only) allowed: double Default: 3.0
stddev	Standard deviation of the noise signal in the image (used by the window and fit functions only) allowed: double Default: 0.0
doppler	Velocity doppler definition for velocity computations

Returns
image

Example

```
"""
#
print "\t----\t moments Ex 1 \t----"
ia.fromshape(shape=[32,32,32,32]) # replace with your own cube
im2 = ia.moments(moments=[-1,1,2], axis=2, smoothaxes=[0,1,2],
                 smoothtypes=["gauss","gauss","hann"],
                 smoothwidths=[5.0,5.0,3], excludepix=[1e-3],
                 smoothout='smooth', overwrite=true)
im2.done()
ia.close()
#
"""
```

In this example, standard moments (average intensity, weighted velocity and weighted velocity dispersion) are computed via the convolve (spatially convolved by gaussians and spectrally by a Hanning kernel) and clip method (we exclude any pixels with absolute value less than 0.001). The output file names are automatically created for us and the convolved image is saved. The returned image tool holds the first moment image.

Example

```
"""
#
print "\t----\t moments Ex 2 \t----"
ia.fromshape(shape=[32,32,32,32])
im2 = ia.moments(moments=[3], method=["window"])
im2.done()
```

```
ia.close()
#
"""
```

In this example, the median of each spectrum is computed, after pixel selection by the automatic window method. The output image is temporary and accessed via the returned Image tool.

---

image.name.html

image.name - Function

1.1.1 Name of the image file this tool is attached to

Description

This function returns the name of the `image file` By default, this function returns the full absolute path of the `image file`. You can strip this path off if you wish with the `strippath` argument and just recover the `image file` name itself.

Arguments

Inputs		
strippath	Strip off the path before the actual file name?	
	allowed:	bool
	Default:	false

Returns

string

Example

```
"""
#
print "\t----\t name Ex 1 \t----"
ia.maketestimage('g1.app', overwrite=true)
print ia.name(strippath=F)
#/casa/code/xmlcasa/implement/images/scripts/g1.app
print ia.name(strippath=T)
#g1.app
ia.close()
#
"""
```

image.open.html

image.open - Function

1.1.1 Open a new image file with this image tool

Description

Use this function when you are finished analyzing the current `image file` and want to attach to another one. This function detaches the `image tool` from the current `image file`, and reattaches it (opens) to the new `image file`. The input image file may be in native CASA, FITS, or Miriad format. Look here for more information on foreign images. In the case of CASAimages, both Float and Complex valued images are supported.

Arguments

Inputs		
infile	image file name	
	allowed:	string
	Default:	

Returns
bool

Example

```
"""
#
print "\t----\t open Ex 1 \t----"
ia.maketestimage('anotherimage',overwrite=true) #first make 2nd image
ia.close()
ia.maketestimage('myimage',overwrite=true)      #open image myimage
ia.open('anotherimage')               # attach tool to 'anotherimage'
ia.close()
#
"""
```

```
The {\stff open} function first closes the old \imagefile.
```

---

image.pad.html

image.pad - Function

1.1.1 Pad the perimeter of the direction plane with a number of pixels of specified value and mask.

Description

This method pads the directional plane of an image with a specified number of pixels on each side. The numerical and mask values of the padding pixels may also be specified. If a region is selected, a subimage of that region is created and then padded with the specified pixel parameters. Thus, padding an image of shape (ra, dec, freq) = (512, 512, 10) specifying npixels = 3 results in an image of size (518, 518, 10), with the blc of the directional plane of the original pixel set corresponding to the directional pixel of (3, 3) in the output. If wantreturn is True, an image analysis tool attached to the output image is returned. If False, none is returned.

Arguments

Inputs	
outfile	Output image name. If not specified, no persistent image is created. allowed: string Default:
npixels	Number of pixels with which to pad each side of the direction plane. allowed: int Default: 1
value	Value given to the padding pixels. allowed: double Default: 0
padmask	Value of the mask for the padding pixels. True=>good (unmasked), False=>bad (masked). allowed: bool Default: false
overwrite	Overwrite the output if it exists? Default False allowed: bool Default: false
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
box	Rectangular region to select in direction plane. See "help par.box" for details. Default is to use the entire direction plane. allowed: string Default:
chans	Channels to use. See "help par.chans" for details. Default is to use all channels. allowed: string Default:
stokes	Stokes planes to use. See "help par.stokes" for details. Default is to use all stokes planes. allowed: string Default:
mask	Mask to use. See help par.mask. Default is none. allowed: string Default:
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false
wantreturn	Return an image analysis tool attached to the created subimage? allowed: bool Default: true

Returns
image

Example

```
ia.fromshape("",[50, 50, 10])
# pad it with 5 pixels of value 2.5 all unmasked
padded = ia.pad(npixels=5, value=2.5, padmask=True)
ia.done()
# returns [60, 60, 10]
paddedshape = padded.shape()
padded.done()
```

---

image.crop.html

image.crop - Function

1.1.1 Crop masked pixels from the perimeter of an image.

Description

This method crops masked slices from the perimeter of an image. The axes parameter specifies which axes to consider. Axes not specified will not be cropped. An empty array implies that all axes should be considered. If wantreturn is True, an image analysis tool attached to the output image is returned. If False, none is returned.

Arguments

Inputs	
outfile	Output image name. If not specified, no persistent image is created. allowed: string Default:
axes	Axes to crop. Empty array means consider all axes. allowed: intArray Default:
overwrite	Overwrite the output if it exists? Default False allowed: bool Default: false
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
box	Rectangular region to select in direction plane. See "help par.box" for details. Default is to use the entire direction plane. allowed: string Default:
chans	Channels to use. See "help par.chans" for details. Default is to use all channels. allowed: string Default:
stokes	Polarization selection. Default is all. allowed: string Default:
mask	Mask to use. See help par.mask. Default is none. allowed: string Default:
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false
wantreturn	Return an image analysis tool attached to the created subimage? allowed: bool Default: true

Returns
image

Example

```
# myimage is of shape 20, 20, 20 with only the inner 16 x 14 x 12 pixels unmasked
ia.open("myimage")
# crop masked slices on all axes
cropped = ia.crop()
# returns [16, 14, 12]
cropped.shape()
cropped.done()
# crop only the masked slices at the edges of the image along axis 1
cropped2 = ia.crop(outfile="", axes=[1])
ia.done()
# returns [20, 14, 20]
cropped2.shape()
cropped2.done()
```

---

image.pixelvalue.html

image.pixelvalue - Function

1.1.1 Get value of image and mask at specified pixel coordinate

Description

This function gets the value of the image and the mask at the specified pixel coordinate. The values are returned in a record with fields 'value', 'mask' and 'pixel'. The value is returned as a quantity, the mask as a Bool (T is good). The 'pixel' field holds the actual pixel coordinate used.
If the specified pixel coordinate is off the image, "{}" is returned.
Excessive elements in `pixel` are silently discarded. Missing elements are given the (nearest integer) value of the reference pixel. This is reflected in the output record 'pixel' field.

Arguments

Inputs		
pixel	Pixel coordinate	
	allowed:	intArray
	Default:	-1

Returns
record

Example

```
"""
#
print "\t----\t pixelvalue Ex 1 \t----"
ia.maketestimage();
ia.pixelvalue()
#{'mask': True,
# 'pixel': array([55, 37]),
# 'value': {'unit': 'Jy/beam', 'value': 2.5064315795898438}}
```

```
print ia.pixelvalue([-1,-1])
# {}
print ia.pixelvalue([9])
#{'mask': True,
# 'pixel': array([ 9, 37]),
# 'value': {'unit': 'Jy/beam', 'value': 0.14012207090854645}}
print ia.pixelvalue([9,9,9])
#{'mask': True,
# 'pixel': array([9, 9]),
# 'value': {'unit': 'Jy/beam', 'value': -0.45252728462219238}}
ia.close()
#
"""
```

---

image.putchunk.html

image.putchunk - Function

1.1.1 Put pixels from an array into a regular region of the image

Description

This function puts an array into the `image file`. If there is a default `pixel mask` it is ignored in this process. It is the complement of the getchunk function. You can specify the `blc` and `inc` if desired. If they are unspecified, they default to the beginning of the image and an increment of one.
Any illegal `blc` values are set to zero. Any illegal `inc` values are set to unity. An error will result if you attempt to put an array beyond the extent of the image (i.e., it is not truncated or decimated).
If there are fewer axes in the array than in the image, the array is assumed to have trailing axes of length unity. Thus, if you have a 2D array and want to put it in as the YZ plane rather than the XY plane, you must ensure that the shape of the array is [1,nx,ny].
However, the argument `replicate` can be used to replicate the array throughout the image (from the blc to the trc). For example, if you provide a 2D array to a 3D image, you can replicate it through the third axis by setting `replicate=T`. The replication is done from the specified `blc` to the end of the image. Use function putregion if you want to terminate the replication at a `trc` value.
The argument `locking` controls two things. If True, then after the function is called, the image is unlocked (so some other process can acquire a lock) and it is indicated that the image has changed. The reason for having this argument is that the unlocking and updating processes are quite expensive. If you are repeatedly calling `putchunk` in a for loop, you would be advised to use this switch.
A related function is putregion which puts the pixels and masks into a more complex `region-of-interest`. Function `putchunk` is retained because it is faster and therefore preferable for repeated operation in loops if the `pixel mask` is not required.
See also the functions set and calc which can also change pixel values.

Arguments

Inputs		
pixels	Numeric array. Required input.	
	allowed:	any
	Default:	variant
blc	Bottom-Left-Corner (start) of location in image. Default is start of image.	
	allowed:	intArray
	Default:	-1
inc	increment (stride) along axes	
	allowed:	intArray
	Default:	1
list	List bounding box to logger?	
	allowed:	bool
	Default:	false
locking	Unlock image after use?	
	allowed:	bool
	Default:	true
replicate	Replicate array through image	
	allowed:	bool
	Default:	false

Returns
bool

Example

We can clip all pixels to be {\tt <= } 5 as follows.

```
"""
#
print "\t----\t putchunk Ex 1 \t----"
ia.fromshape(shape=[10,10])   # create an example image
pix = ia.getchunk()           # get pixels to modify from example image
for i in range(len(pix)):
  pix[i] = list(pix[i])       # convert tuple to list so it can be modified
  for j in range(len(pix[i])):
    pix[i][j] = i*10 + j
  pix[i] = tuple(pix[i])      # convert list back to tuple
ia.putchunk(pix)              # put pixels back into example image
print pix                     # pixels have values 0-99
```

```
pix2 = ia.getchunk()          # get all pixels into an array (again)
for i in range(len(pix2)):
  pix2[i] = list(pix2[i])     # convert tuple to list so it can be modified
  for j in range(len(pix2[i])):
    if pix2[i][j] > 5:
      pix2[i][j] = 5          # clip values to 5
  pix2[i] = tuple(pix2[i])    # convert list back to tuple
ia.putchunk(pix2)             # put array back into image
print ia.getchunk()
ia.close()
#
"""
```

The above example shows how you could clip an image to a value. If all the pixels didn't easily fit in memory, you would iterate through the image chunk by chunk to avoid exhausting virtual memory. Better would be to do this via LEL through function calc.

Suppose we wanted to set the fifth XY plane to 1.

We could do so as follows:

```
"""
#
print "\t----\t putchunk Ex 2 \t----"
ia.fromshape(shape=[10,10,10])
imshape = ia.shape()
pix = ia.makearray(1, [imshape[0],imshape[1]])
ia.putchunk(pix, blc=[0,0,4])
print ia.getchunk()[0:3]
ia.close()
#
"""
```

Suppose we wanted to set the first YZ plane to 2.

```
"""
#
print "\t----\t putchunk Ex 3 \t----"
ia.fromshape(shape=[10,10,10])
imshape = ia.shape()
pix = ia.makearray(2, [1,imshape[1],imshape[2]])
ia.putchunk(pix)
```

```
print ia.getchunk()[0:3]
ia.close()
#
"""
```

---

image.putregion.html

image.putregion - Function

1.1.1 Put pixels and mask into a region-of-interest of the image

Description

This function replaces data and/or `pixel mask` values in the image in the specified `region-of-interest`. The `pixels` and/or `pixelmask` arrays must be the shape of the bounding box, and the whole bounding box is replaced in the image. The `region-of-interest` is really only used to specify the bounding box. If the region extends beyond the image, it is truncated. If the `pixels` or `pixelmask` array shapes do not match the bounding box, an error will result. When you put a `pixel mask`, it either replaces the current default `pixel mask`, or is created. The `pixel mask` is put before the pixels.
The argument `usemask` is only relevant when you are putting pixel values and there is a `pixel mask` (meaning also the one you might have just put in place). If `usemask=T` then only pixels for which the mask is good (T) are altered. If `usemask=F` then all the pixels in the region are altered - the mask is ignored.
The argument `replicate` can be used to replicate the array throughout the image (from the blc to the trc). For example, if you provide a 2D array to a 3D image, you can replicate it through the third axis by setting `replicate=T`. The replication is done in the specified `region`.
The argument `locking` controls two things. If True, then after the function is called, the image is unlocked (so some other process can acquire a lock) and it is indicated that the image has changed. The reason for having this argument is that the unlocking and updating processes are quite expensive. If you are repeatedly calling `putregion` in a for loop, you would be advised to use this switch (and to consider using `putchunk`).
See the related functions putchunk, set and calc.

Arguments

Inputs	
pixels	The pixel values. Default is none. allowed: any Default: variant
pixelmask	The pixel mask values. Default is none. allowed: any Default: variant
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: any
list	List the bounding box and any mask creation to the logger allowed: bool Default: false
usemask	Honour the mask when putting pixels allowed: bool Default: true
locking	Unlock image after use? allowed: bool Default: true
replicate	Replicate array through image allowed: bool Default: false

Returns
bool

Example

```
Suppose that we have a 2-dimensional image.  First we recover the pixel
and \pixelmask\ values from a polygonal region.  Then we change the values in
the array that are within the region to zero and replace the data.


"""
#
```

```
print "\t----\t putregion Ex 1 \t----"
ia.maketestimage()                          # Attach an image to image tool
x = ['3pix','6pix','9pix','6pix','5pix','5pix','3pix'] # X vector abs pixels
y = ['3pix','4pix','7pix','9pix','7pix','5pix','3pix'] # Y vector abs pixels
mycs = ia.coordsys()
r1 = rg.wpolygon(x,y,csys=mycs.torecord()) # Create polygonal world region
mycs.done()
pixels = ia.getregion(r1)                   # Recover pixels
pixelmask = ia.getregion(r1, getmask=T)     # and mask
for i in range(len(pixels)):
  pixels[i] = list(pixels[i])               # convert tuple to list for mods
  for j in range(len(pixels[i])):
    if pixelmask[i][j]:
      pixels[i][j] = 0                      # Set pixels where mask is T to zero
  pixels[i] = tuple(pixels[i])              # convert list back to tuple
ia.putregion(pixels=pixels, pixelmask=pixelmask,
             region=r1)                     # Replace pixels only
ia.close()
#
"""
```

---

image.rebin.html

image.rebin - Function

1.1.1 Rebin an image by the specified integer factors

Description

This application rebins the current image by the specified integer binning factors for each axis. It supports both float valued and complex valued images. The corresponding output pixel value is the average of the input pixel values. The output pixel will be masked bad if there were no good input pixels. A polarization axis cannot be rebinned.
The binning factors array must contain at least one element and no more elements than the number of input image axes. If the number of elements specified is less than the number of image axes, then the remaining axes not specified are not rebinned. All specified values must be positive. A value of one indicates that no rebinning of the associated axis will occur.
Binning starts from the origin pixel of the bounding box of the selected region or the origin pixel of the input image if no region is specified. The value of crop is used to determine how to handle cases where there are pixels at the end of the axis that do not form a complete bin. If crop=True, extra pixels at the end of the axis are discarded. If crop=False, the remaining pixels are averaged into the final bin along that axis. Should the length of the axis to be rebinned be an integral multiple of the associated binning factor, the value of crop is irrelevant.
A value of dropdeg=True will result in the output image not containing axes that are degenerate in the specified region or in the input image if no region is specified. Note that, however, the binning factors array must still account for degenerate axes, and the binning factor associated with a degenerate axis must always be 1.
If `outfile` is given, the image is written to the specified disk file. If `outfile` is unset, the Image `tool` is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the on-the-fly Image `tool` returned by this function (with the done function) this temporary image is deleted.

Arguments

Inputs	
outfile	Output image file name. Default is unset. allowed: string Default:
bin	Binning factors for each axis allowed: intArray Default:
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
dropdeg	Drop degenerate axes allowed: bool Default: false
overwrite	Overwrite (unprompted) pre-existing output file? allowed: bool Default: false
async	Run asynchronously? allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false
crop	Remove pixels from the end of an axis to be rebinned if there are not enough to form an integral bin? allowed: bool Default: false

Returns
image

Example

"""

```
#
print "\t----\t rebin Ex 1 \t----"
ia.maketestimage();
im2 = ia.rebin(bin=[2,3]);
im2.done()
ia.close()
#
"""
```

---

image.regrid.html

image.regrid - Function

1.1.1 regrid this image to the specified Coordinate System

Description

This function regrids the current image onto a grid specified by the given Coordinate System. You can also specify the shape of the output image.
The Coordinate System must be given via a Coordsys `tool` (using coordsys.torecord()). It is optional; if not specified, the Coordinate System from the input image (i.e. the one to which you are applying the regrid function) is taken. The order of the coordinates and axes in the output image is always the same as the input image. It simply 'finds' the relevant coordinate in the supplied Coordinate System in order to figure out the regridding parameters. The supplied Coordinate System must have at least as many coordinates as are required to accomodate the axes you are regridding (e.g. if you regrid the first two axes, and these belong to a Direction Coordinate, you need one Direction Coordinate in the supplied Coordinate System).
Coordinates pertaining to axes that are not being regridded are supplied from the input image, not the given Coordinate System.
Reference changes are handled (e.g. J2000 to B1950, LSR to TOPO). In general, the conversion machinery attempts to work out how sophisticated it needs to be (e.g. am I regridding LSR to LSR or LSR to TOPO). However, it errs on the side of conservatism so that it can be that the conversion machine requires more information than it actually needs. For full frame conversions, one needs to know things like location on earth (e.g. observatory), direction of observation, and time of observation.
If you get the above errors and you **are** doing a frame conversion, then that means you must insert some extra information into the Coordinate System of your image. Most likely it's the time (coordsys.setepoch) and location (coordsys.settelescope) that are missing. If you get these errors and you **know** that you are not specifying a frame change (e.g. regrid LSR to LSR) then try setting `doref=F`. This will (silently) bypass all possible frame conversions.
Note that if you **are** requesting a frame conversion and you set `doref=F` you are doing a bad thing (and you will get no warnings).
If you regrid a plane holding a Direction Coordinate and the units are Jy/pixel then the output is scaled to conserve flux (roughly; just one scale factor at the reference pixel is computed).
Regridding of complex-valued images is supported. The real and imaginary parts are regridded independently and the resulting regridded pixel values are combined to form the regridded, complex-valued image.

A variety of interpolation schemes are provided (you need only specify the first three characters to `method`). The cubic interpolation is substantially slower than linear, and often the improvement is modest. By default you get linear interpolation.
You specify the shape of the output image (`shape`) and which output axes you want to regrid (`axes`). Note that a Stokes axis cannot be regridded (you will get a warning if you try).
The `axes` argument cannot be used to discard axes from the output image; it can only be used to specify which **output** axes are going to be regridded and which are not. Any axis that you are not regridding must have the same output shape as the input image shape for that axis.
The `axes` argument can also be used to specify the order in which the **output** axes are regridded. This may give you significant performance benefits. For example, imagine we are going to regrid a spectral-line cube of shape [512,512,1204] to shape [256,256,32]. If you specified `axes=[0,1,2]` then first, the Direction axes would be regridded for each of the 1024 pixels (and stored in a temporary image). Then each profile at each spatial location in the temporary image would be regridded to 32 pixels. You could speed this process up significantly by setting `axes=[2,0,1]`. In this case, first each profile would be regridded to 32 pixels, and then each plane of the 32 pixels would be regridded. Note that the order of `axes` does not affect the order of the `shape` argument. I.e. it should be given in the natural pixel axis order of the image `[256,256,32]` in both cases.
You can also specify a `region-of-interest` to be applied to the input image. If you do this, you need to be careful with the output shape for non-regridded axes (must match that of the region - use function boundingbox to find that out).
If `outfile` is given, the image is written to the specified disk file. If `outfile` is unset, the on-the-fly Image `tool` returned by this function is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the on-the-fly Image `tool` (with the done function) this temporary image is deleted.
The argument `replicate` can be used to simply replicate pixels rather than regridding them. Normally (`replicate=F`), for every output pixel, its world coordinate is computed and the corresponding input pixel found (then a little interpolation grid is generated). If you set `replicate=T`, then what happens is that for every output axis, a vector of regularly sampled input pixels is generated (based on the ratio of the output and input axis shapes). So this just means the pixels get replicated (by whatever interpolation scheme you use) rather than regridded in world coordinate space. This process is much faster, but its not a true world coordinate based regrid.
As decribed above, when `replicate` is False, a coordinate is computed for each output pixel; this is an expensive operation. The argument `decimate` allows you to decimate the computation of that coordinate grid to a sparse grid, which is then filled in via fast interpolation. The default for `decimate` is 10. The number of pixels per axis in the sparse grid is the number of output

pixels for that axis divided by the decimation factor. A factor of 10 does pretty well. You may find that for very non-linear coordinate systems (e.g. very close to the pole) that you have to reduce the decimation factor. You may also have to reduce the decimation factor if the number of pixels in the output image along an axis to be regridded is less than about 50, or the output image may be completely masked.
If one of the axes to be regridded is a spectral axis and asvelocity=T, the axis will be regridded to match the velocity, not the frequency, description of the template coordinate system. Thus the output pixel values will correspond only to the velocity, not the frequency, of the output axis.
Sometimes it is useful to drop axes of length one (degenerate axes). Use the `dropdeg` argument if you want to do this. It will discard the axes from the input image. Therefore the output shape and Coordinate System that you supply must be consistent with the input image after the degenerate axes are dropped.
Argument `force` can be used to force all specified axes to be regridded, even if the algorithm determines that they don't need to be (because the input and output coordinate information is identical).
There is a useful function setreferencelocation that you can use to keep a specific world coordinate in the center of an image when regridding (see example below).
The output `pixel mask` will be good (T) unless the regridding failed to find a value for that output pixel in which case it will be bad (F). For example, if the total input mask (default input `pixel mask` plus OTF mask) for all of the relevant input pixels were masked bad then the output pixel would be masked bad (F).
Multiple axis Coordinates limitation – Some cooordinates pertain to more than one axis. E.g. a Direction Coordinate holds longitude and latitude. A Linear Coordinate can also hold many axes. When you regrid *any* axis from a Coordinate which holds multiple axes, you must fully specify the coordinate information for all axes in that Coordinate in the Coordinate System that you provide. For example, you have a Linear Coordinate with two axes and you want to regrid axis one only. In the Coordinate System you provide, the coordinate information for axis two (not being regridded) must correctly be a copy from the input coordinate system (it won't be filled in for you).
If an image has per-plane beams and one attempts to regrid the spectral axis, an exception is thrown.
IMPORTANT NOTE ABOUT FLUX CONSERVATION in general regridding is inaccurate for images that the angular resolution is poorly sampled. A check is done for such cases and a warning message is emitted if a beam present. However, no such check is done if there is no beam present. To add a restoring beam to an image, use ia.setrestoringbeam().

Arguments

Inputs	
outfile	Output image file name. Default is unset. allowed: string Default:
shape	Shape of output image. Default is input shape. allowed: intArray Default: -1
csys	Coordinate System for output image. Default is input image coordinate system. allowed: record Default:
axes	The output pixel axes to regrid. Default is all. allowed: intArray Default: -1
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
method	The interpolation method. String from 'nearest', 'linear', 'cubic'. allowed: string Default: linear
decimate	Decimation factor for coordinate grid computation allowed: int Default: 10
replicate	Replicate image rather than regrid? allowed: bool Default: false
doref	Turn on reference frame changes allowed: bool Default: true
dropdeg	Drop degenerate axes allowed: bool Default: false
overwrite	Overwrite (unprompted) pre-existing output file? allowed: bool Default: false
force	Force specified axes to be regridded allowed: bool Default: false
asvelocity	Regrid spectral axis in velocity space rather than frequency space? allowed: bool Default: false
async	Run asynchronously? allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false

Returns
image

Example

```
"""
#
print "\t----\t regrid Ex 1 \t----"
ia.maketestimage('radio.image', overwrite=true)
ia.maketestimage('optical.image', overwrite=true)
mycs = ia.coordsys();     # get optical image co-ordinate system
ia.open('radio.image')
imrr = ia.regrid(outfile='radio.regridded', csys=mycs.torecord(),
                  shape=ia.shape(), overwrite=true)
#viewer()
mycs.done()
imrr.done()
ia.close()
#
"""
```

In this example, we regrid a radio image onto the grid of an optical image - this probably (if the optical FITS image was correctly labelled !!) will involve a projection change (optical images are usually TAN projection, radio usually SIN).

Example

```
"""
#
print "\t----\t regrid Ex 2 \t----"
ia.maketestimage('radio.image',overwrite=true)
mycs = ia.coordsys();
print mycs.referencecode('dir')
#J2000
```

```
mycs.setreferencecode(value='B1950', type='dir', adjust=T)
im3 = ia.regrid(outfile='radio.regridded', csys=mycs.torecord(),
                shape=ia.shape(), overwrite=true)
mycs.done()
im3.done()
ia.close()
#
"""
```

In this example, we regrid a radio image from J2000 to B1950. This is accomplished by first recovering the Coordinate System into a Coordsys tool, manipulating the reference code
with that \tool, and then supplying the new Coordinate System to the regrid function.

Example

```
"""
#
print "\t----\t regrid Ex 3 \t----"
ia.maketestimage('zz', overwrite=true)
mycs = ia.coordsys();
p = ia.shape()
for i in range(len(p)):
  p[i] = p[i]/2.0 + 10
refval = ia.toworld(value=p, format='n') # Location of interest
inc = mycs.increment()
incx = inc['numeric']
for i in range(len(incx)):
  incx[i] = incx[i]/2.0                  # Halve increment
inc['numeric']=incx
mycs.setincrement(value=inc)             # Set increment
shp = ia.shape()
refpix=refval['numeric'][:]
refpix=list(refpix)                      # numpy makes this necessary
for i in range(len(shp)):
  shp[i] = shp[i] *2                     # Double shape
  refpix[i] = int((shp[i]-1)/2.0 + 1);   # New ref pix
# Center image on location of interest
mycs.setreferencelocation(pixel=refpix, world=refval)
```

```
imr = ia.regrid(csys=mycs.torecord(), shape=shp, overwrite=true)# Regrid
mycs.done()
imr.done()
ia.close()
#
"""
```

---

image.transpose.html

image.transpose - Function

1.1.1 Transpose the image.

Description

This method transposes the axes in the input image to the specified order. The associated pixel and mask values and coordinate system are transposed. If the outfile parameter is empty, only a temporary image is created; no output image is written to disk.
The order parameter describes the mapping of the input axes to the output axes. It can be one of three types: a non-negative integer, a string, or a list of strings. If a string or non-negative integer, it should contain zero-based digits describing the new order of the input axes. It must contain the same number of (unique) digits as the number of input axes. For example, specifying reorder="1032" or reorder=1032 for a four axes image maps input axes 1, 0, 3, 2 to output axes 0, 1, 2, 3. In the case of order being a nonnegative integer and the zeroth axis in the input being mapped to zeroth axis in the output, the zeroth digit is implicitly understood to be 0 so that to transpose an image where one would use a string order="0321", one could equivalently specify an int order=321. IMPORTANT: When specifying a non-negative integer and mapping the zeroth axis of the input to the zeroth axis of the output, do *not* explicitly specify the leading 0; eg, specify order=321 rather than order=0321. Python interprets an integer with a leading 0 as an octal number.
Because of ambiguity for axes numbers greater than nine, using string or integer order specifications cannot handle images containing more than 10 axes. The order parameter can also be specified as a list of strings which uniquely minimally match, ignoring case, the image axis names (ia.coordsys().names()). So to reorder an image with right ascension, declination, and frequency axes, one could specify order=["d", "f", "r"] or equivalently ["decl", "frequ", "right a"]. Note that specifying "ra" for the right ascension axis will result in an error because "ra" does not match the first two characters of right ascension. Axes can be simultaneously inverted in cases where order is a string or an array of strings by specifying negative signs in front of the axis/axes to be inverted. So, in a 4-D image, order="-10-3-2" maps input axes 1, 0, 3, 2 to output axes 0, 1, 2, 3 and reverses the direction and values of input axes 1, 3, and 2.

Arguments

Inputs		
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	
order	Zero-based order of axes in output image (eg "120" => input− > output 0->2, 1->0, 2->1))	
	allowed:	any
	Default:	variant

Returns
image

Example

```
"""
# swap stokes (axis 2) and spectral (axis 3) axes in a 4 dimensional image
ia.open("myimage.fits")
reordim = ia.transpose(outfile="my_reordered_image.im", order="0132")
ia.done()
"""
```

---

image.rotate.html

image.rotate - Function

1.1.1 rotate the direction coordinate axes attached to the image and regrid the image to the rotated Coordinate System

Description

This function rotates two axes of an image. These axes are either those associated with a Direction coordinate or with a Linear coordinate. The Direction coordinate takes precedence. If rotating a Linear coordinate, it must hold precisely two axes.
The method is that the Coordinate is rotated and then the input image is regridded to the rotated Coordinate System.
If the image brightness units are Jy/pixel then the output is scaled to conserve flux (roughly; just one scale factor at the reference pixel is computed).
A variety of interpolation schemes are provided (you need only specify the first three characters to `method`). The cubic interpolation is substantially slower than linear. By default you get cubic interpolation.
You can specify the shape of the output image (`shape`). However, all axis that are not regrided retain the same output shape as the input image shape for that axis. Only the direction coordinate axes are regridded.
You can also specify a `region-of-interest` to be applied to the input image. If you do this, you need to be careful with the output shape for non-regridded axes (must match that of the region - use function boundingbox to find that out).
If `outfile` is given, the image is written to the specified disk file. If `outfile` is unset, the on-the-fly Image `tool` returned by this function is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the on-the-fly Image `tool` (with the done function) this temporary image is deleted.
The argument `replicate` can be used to simply replicate pixels rather than regridding them. Normally (`replicate=F`), for every output pixel, its world coordinate is computed and the corresponding input pixel found (then a little interpolation grid is generated). If you set `replicate=T`, then what happens is that for every output axis, a vector of regularly sampled input pixels is generated (based on the ratio of the output and input axis shapes). So this just means the pixels get replicated (by whatever interpolation scheme you use) rather than regridded in world coordinate space. This process is much faster, but its not a true world coordinate based regrid.
As decribed above, when `replicate` is False, a coordinate is computed for each output pixel; this is an expensive operation. The argument `decimate` allows

you to decimate the computation of that coordinate grid to a sparse grid, which is then filled in via fast interpolation. The default for `decimate` is 0 (no decimation). The number of pixels per axis in the sparse grid is the number of output pixels for that axis divided by the decimation factor. A factor of 10 does pretty well. You may find that for very non-linear coordinate systems (e.g. very close to the pole) that you have to reduce the decimation factor. The output `pixel mask` will be good (T) unless the regridding failed to find a value for that output pixel in which case it will be bad (F). For example, if the total input mask (default input `pixel mask` plus OTF mask) for all of the relevant input pixels were masked bad then the output pixel would be masked bad (F).

Arguments

Inputs	
outfile	Output image file name. Default is unset. allowed: string Default:
shape	Shape of output image. Default is shape of input image. allowed: intArray Default: -1
pa	Angle by which to rotate. Default is no rotation. allowed: any Default: variant 0deg
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
method	The interpolation method. String from 'nearest', 'linear', or 'cubic'. allowed: string Default: cubic
decimate	Decimation factor for coordinate grid computation allowed: int Default: 0
replicate	Replicate image rather than regrid? allowed: bool Default: false
dropdeg	Drop degenerate axes allowed: bool Default: false
overwrite	Overwrite (unprompted) pre-existing output file? allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false

Returns
image

Example

```
"""
ia.maketestimage()
imr=ia.rotate(outfile="rotated.im", pa='45deg')
imr.done()
ia.close()
```

In this example, we rotate the direction coordinate axes (RA/Dec) of a test image by 45 degress and regrid the image onto the axes.

---

image.rotatebeam.html

image.rotatebeam - Function

1.1.1 rotate the image's beam(s) counterclockwise through the specified angle.

Description

This method rotates the attached image's beam(s) counterclockwise through the specified angle. This is the same thing as increasing the position angle(s) of the beam(s) by the specified angle. If the image does not have a beam, no changes to the image are made. If the image has multiple beams, all the beams are rotated through the same angle.

Arguments

Inputs		
angle	Angle by which to rotate image's beam(s). Default is no rotation.	
	allowed:	any
	Default:	variant 0deg

Returns
bool

Example

```
# rotate any and all beams in the image (increase their position angles) by 30 degrees.
ia.open("my.im")
ia.rotatebeam("30deg")
ia.done()
```

image.rename.html

image.rename - Function

1.1.1 Rename the image file associated with this image tool

Description

This function renames the `image file` associated with the `image tool`. If a file with name `name` already exists, you can overwrite it with the argument `overwrite`; otherwise a fail will result.

Arguments

Inputs		
name	The new image file name	
	allowed:	string
	Default:	
overwrite	Overwrite target file if it already exists	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
"""
#
print "\t----\t rename Ex 1 \t----"
ia.maketestimage('myimage',overwrite=T)
print ia.name(strippath=T)
#myimage
ia.rename('newimage', overwrite=T)
print ia.name(strippath=T)
#newimage
#
```

"""

---

image.replacemaskedpixels.html

image.replacemaskedpixels - Function

1.1.1 replace the values of pixels which are masked bad

Description

This application replaces the values of all pixels whose total input mask (default input `pixel mask` and OTF mask) is bad (F) with the specified value. It supports both float valued and compplex valued images.
If the argument `update` is F (the default), the actual `pixel mask` is left unchanged. That is, masked pixels remain masked. However, if you set `update=T` then the `pixel mask` will be updated so that the `pixel mask` will now be T (good) where the **total** input mask was F (bad).
See maskhandler for information on how to set the default `pixel mask`.
There are a few ways in which you can specify what to replace the masked pixel values by.

- First, you can give the `pixels` argument a simple numeric scalar (e.g. `pixels=1.0`). Then, all masked values will be replaced by that value.
- Second, you can give a scalar LEL expression string (e.g. `pixels='min(myimage)'`). Then, all masked values will be replaced by the scalar that results from the expression. If the scalar expression is illegal (e.g. in the expression `pixels='min(myimage)'` there were no good pixels in *myimage*) then the value 0 is used for replacement.
- Third, you can give a LEL expression string which has the same shape as the `image file` you are applying the function to. For example, putting `pixels='myotherimage'` means replace all masked pixels in this `image file` with the equivalent pixel in the `image file` called *myotherimage*.

  Your expression might be quite complex, and you can think of it as producing another masked lattice. However, in the replace process, the mask of that expression lattice is ignored. Thus, only the mask of the `image file` you are replacing and the pixel values of the expression lattice are relevant.

  The expression must conform with the subimage formed by applying the `region-of-interest` to the image (i.e. that associated with this Image `tool`). If you use the `mask` argument as well, the `region-of-interest` is applied to it as well (see examples).

Arguments

Inputs	
pixels	The new value(s), Numeric scalar or LEL expression allowed: any Default: variant
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
update	Update mask as well? allowed: bool Default: false
list	List the bounding box to the logger allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false

Returns
bool

Example

```
"""
#
print "\t----\t replacemaskedpixels Ex 1 \t----"
ia.maketestimage('zz1',overwrite=true)
ia.calcmask('zz1<0')
ia.replacemaskedpixels(0.0)
ia.replacemaskedpixels('min(zz1)')
ia.close()
#
"""
```

These examples replace all masked pixels by the specified scalar. In the second case, the scalar comes from a LEL expression operating on {\sff zz1} (or it could be from an LEL expression operating on some other image).

Example

```
"""
#
print "\t----\t replacemaskedpixels Ex 2 \t----"
ia.maketestimage('zz2',overwrite=true)
ia.close()
ia.maketestimage('zz1',overwrite=true)
#ia.calcmask('zz1<0')
ia.replacemaskedpixels(0.0, mask='zz2>0')
ia.close()
#
"""
```

Let us say that {\sff zz1} has no mask. By using the {\stfaf mask} argument, we generate a transient mask which is T (good) when the pixel values are positive. This means that all non-positive values (when that mask is F [bad]) will be replaced with the value 0. If {\sff zz1} did have a mask it would be applied as well as the transient mask (the masks would be logically ORed).

Example

```
"""
#
print "\t----\t replacemaskedpixels Ex 3 \t----"
```

```
ia.maketestimage('zz1',overwrite=true)
ia.calcmask('zz1<0')
im2 = ia.subimage(outfile='zz2',overwrite=true)
# r = rg.quarter()
r=rg.box([0.25,0.25],[0.75,0.75],frac=true)
ia.replacemaskedpixels(0.0, region=r, mask=im2.name(strippath=T)+'>0')
# same as ia.replacemaskedpixels(0.0, region=r, mask='zz2>0')
im2.done()
ia.close()
#
"""
```

The specified region takes one quarter of the image by area centered on the image center. The region is applied to the {\stfaf mask} expression as well - this means that any images in the {\stfaf mask} expression must conform with the {\sff zz1} image. The replacement of the scalar is then done only within that region. Note that in the {\stfaf mask} expression we have specified the image with the Image tool {\stf im2} via im2.name() (rather than referring to its disk file name {\sff zz2}).

Example

```
#
print "\----\t replacemaskedpixels Ex 4 \t----"
ia.maketestimage('zz3',overwrite=true)
ia.maketestimage('zz2',overwrite=true)
ia.maketestimage('zz1',overwrite=true)
ia.calcmask('zz1<0')
ia.replacemaskedpixels('zz2+zz3')
ia.close()
#
```

In this example, the replacement values are taken from a LEL expression adding two other images together. The expression must conform with the

```
image {\sff zz1}.
```

---

image.restoringbeam.html

image.restoringbeam - Function

1.1.1 Get the restoring beam(s).

Description

This function gets the restoring beam(s), if any. If the image has a traditional restoring beam, that is returned no matter what channel and polarization are set to. If the image has per-plane beams and at least one of channel or polarization is set to a non-negative value, the beam for that particular plane is returned. In both these cases, the returned record contains fields 'major', 'minor' and 'postionangle'. Each of these fields contains a quantity.
If the image contains multiple beams and both channel and polarization are negative, a record containing all the beams is returned. This record contains three fields. "nChannels" contains an integer value equal to the number of channels, "nStokes" contains an integer value equal to the number of polarizations, and "beams" contains a record of information for all beams. If the image has no polarization axis or no spectral axis, the fields in the "beams" record run from "*0" to the number of spectral channels or number of polarizations less one, eg "*31" for an image with 32 channels. Each of these fields references a beam subrecord with the structure described above for a single beam.
If the image contains both a spectral axis and a polarization axis, the record returned contains fields running from "*0" to the number of spectral channels less one, eg "*31" for an image with 32 spectral channels. Each of these fields has an associated subrecord with fields running from "*0" to the number of polarizations less one, eg "*3" in an image with 4 polarizations. Each one of those fields is associated with a beam record for that corresponding channel and polarization. The beam record has a structure described above for a single beam.
If there is no restoring beam, this function returds an empty record.
You can set the restoring beam with function setrestoringbeam.

Arguments

Inputs	
channel	The zero-based spectral channel number for a per-plane beam. Default -1 allowed: int Default: -1
polarization	The zero-based polarization plane number for a per-plane beam. Default -1 allowed: int Default: -1

Returns
record

Example

```
"""
#
print "\t----\t restoringbeam Ex 1 \t----"
ia.maketestimage()
print ia.restoringbeam()
#{'major': {'unit': 'arcsec', 'value': 53.500004857778549},
# 'minor': {'unit': 'arcsec', 'value': 34.199998900294304},
# 'positionangle': {'unit': 'deg', 'value': 6.0}}
ia.close()
#
"""
```

image.sepconvolve.html

image.sepconvolve - Function

1.1.1 Separable convolution

Description

This function does Fourier-based convolution of an `image file` by a specified separable kernel.
If `outfile` is given, the image is written to the specified disk file. If `outfile` is unset, the on-the-fly Image `tool` returned by this function is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the Image `tool` (with the done function) this temporary image is deleted.
You specify which axes of the image you wish to convolve, by what kernel of what width. The kernel types can be shortened to `'gauss', 'hann'` and `'box'`.
You specify the widths of the convolution kernels via the argument `widths`. The values can be specified as a vector of three different types.

- Quantity - for example `widths=qa.quantity("1arcsec 0.00001rad")`. Note that you can use pixel units, viz. `widths=qa.quantity("10pix 0.00001rad")` see below.
- String - for example `widths="1km 2arcsec"` (i.e. a string that qa.quantity() accepts).
- Numeric - for example `widths=[10,20]`. In this case, the units of the widths are assumed to be in pixels.

The interpretation of `widths` depends upon the kernel type.

- Gaussian - the specified width is the full-width at half-maximum.
- Boxcar (tophat) - the specified width is the full width.
- Hanning - The kernel is $z[i] = 0.25*y[i-1] + 0.5*y[i] + 0.25*y[i+1]$. The width is always 3 pixels, regardless of what you give (but you still have to give it !).

The scaling of the output image is determined by the argument `scale`. If you leave it unset, then autoscaling will be invoked which means that the convolution kernels will all be normalized to have unit volume to as to conserve flux.

If you do not leave `scale` unset, then the convolution kernel will be scaled by this value (it has peak unity before application of this scale factor).
Masked pixels will be assigned the value 0.0 before convolution. The output mask is the combination (logical OR) of the default input `pixel mask` (if any) and the OTF mask. Any other input `pixel masks` will not be copied. Use function maskhandler if you need to copy other masks too.
See also the other convolution functions convolve2d, convolve and hanning.

Arguments

Inputs	
outfile	Output image file name. Default is unset. allowed: string Default:
axes	Axes to convolve. Default is [0,1,...]. allowed: intArray Default: -1
types	Type of convolution kernel. Vector of strings from 'boxcar', 'gaussian', and 'hanning'. Default is appropriately sized vector of 'gaussian'. allowed: stringArray Default:
widths	Convolution kernel widths, Vector of numeric, quantity or string allowed: any Default: variant
scale	Scale factor. Default is autoscale. allowed: double Default: -1
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: region
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
overwrite	Overwrite (unprompted) pre-existing output file? allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false

Returns
image

Example

```
"""
#
print "\t----\t sepconvolve Ex 1 \t----"
ia.maketestimage('xyv',overwrite=true)
im2 = ia.sepconvolve(outfile='xyv.con', axes=[0,1], types=["gauss","box"], widths=[10,20], c
im2.done()
ia.close()
#
"""
```

---

image.set.html

image.set - Function

1.1.1 Set pixel and/or mask values with a scalar in a region-of-interest of the image

Description

This function replaces data and/or mask values within the image in the specified `region-of-interest`. You can think of it as a simplified version of the image calculator.
Unlike the calc function, you can only set a scalar value for all pixels in the specified `region-of-interest`. For example, it can be useful to set a whole image to one value, or a mask in a `region-of-interest` to one value. Although you could do that with the related functions putregion and putchunk, you would have to make an array of the shape of the image and if that is large, it could be resource expensive.
The value for the pixels is specified with the `pixels` argument. It can be given as either a Lattice Expression Language (or LEL) expression string or a simple numeric scalar. See note 223 for a detailed description of the LEL expression syntax. If you give a LEL expression it must be a scalar expression.
Note that any default mask is *ignored* by this function when you set pixel values. This is different from calc where the extant mask is honoured.
The value for the pixel mask is specified with the `pixelmask` argument (`T, F, unset`). If it's `unset` then the mask is not changed.
If you specify `pixelmask=` T or F, then the mask that is affected is the current default mask (see maskhandler). If there is no mask, a mask is created for you and made the default mask.

Arguments

Inputs	
pixels	The pixel value, LEL scalar expression or numeric scalar. Default is unset. allowed: variant Default:
pixelmask	The pixel mask value. Either 0 or 1 if set. Default is unset. allowed: int Default: -1
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
list	List the bounding box and any mask creation to the logger allowed: bool Default: false

Returns
bool

Example

```
"""
#
print "\t----\t set Ex 1 \t----"
ia.maketestimage('yy',overwrite=true)
ia.fromshape('xx', [10,20], overwrite=true)
r1 = rg.box([2,2],[6,8])         # Make a box region
ia.set(pixels=1.0)               # Set all pixels to 1
ia.set(pixels='2.0', region=r1)  # Set all pixels to 2 in the region
ia.set(pixels='min(yy)')         # Set all pixels to minimum of image yy
                                 # Set pixels in region to minimum of image xx
ia.set(pixels='min('+ia.name(strippath=T)+')', region=r1)
ia.set(pixelmask=T)              # Set mask to all T
ia.set(pixels=0, pixelmask=F, region=r1)  #Set pixels and mask in region
ia.close()
#
"""
```

image.setbrightnessunit.html

image.setbrightnessunit - Function

1.1.1 Set the image brightness unit

Description

This function sets the image brightness unit. Both float and complex valued images are supported. You can get the brightness unit with function brightnessunit.

Arguments

Inputs		
unit	New brightness unit	
	allowed:	string
	Default:	

Returns

bool

Example

```
"""
#
print "\t----\t setbrightnessunit Ex 1 \t----"
ia.fromshape(shape=[10,10])
ia.setbrightnessunit('km')
print ia.brightnessunit()
#km
#
"""
```

image.setcoordsys.html

image.setcoordsys - Function

1.1.1 Set new Coordinate System

Description

This function replaces the coordinate system in the image. It is supported for both float and complex valued images. Coordinate systems are manipulated with the cs `tool`. The coordinate system can be recovered from an image via the coordsys function.
Note that changing the cs `tool` has no effect on the original image, until it is replaced with this function; the value returned by coordsys() is a copy of, not a reference to, the image's coordinate system.

Arguments

Inputs		
csys	Record describing new Coordinate System	
	allowed:	record
	Default:	

Returns
bool

Example

```
"""
#
print "\t----\t setcoordsys Ex 1 \t----"
ia.fromshape(shape=[10,20])          # Make image
mycs = ia.coordsys();                # Recover Coordinate System
incr = mycs.increment('n');          # Get increment as numeric vector
incrn = incr['numeric']
for i in range(len(incrn)):
  incrn[i] = 2*incrn[i]
```

```
incr['numeric']=incrn
mycs.setincrement(value=incr);       # Set new increment in Coordinate System
ia.setcoordsys(mycs.torecord());     # Set new Coordinate System in image
mycs.done()
ia.close()
#
"""
```

---

image.sethistory.html

image.sethistory - Function

1.1.1 Set the history for an image

Description

A CASA `image file` can accumulate history information from an input FITS file or by you writing something into it explicitly with this function. Each element of the input vector is one line of history. The new history is appended to the old.
You can recover the history information with function history.

Arguments

Inputs		
origin	Used to set message origin. Default is image::sethistory.	
	allowed:	string
	Default:	
history	New history	
	allowed:	stringArray
	Default:	

Returns
bool

Example

```
"""
#
print "\t----\t sethistory Ex 1 \t----"
ia.maketestimage('myfile',overwrite=true)
h = ia.history()
# Adds three lines, 'I', 'like' and 'fish'
ia.sethistory(origin="sethistory", history=["I","like","fish"])
ia.close()
```

```
#
"""
```

---

image.setmiscinfo.html

image.setmiscinfo - Function

1.1.1 Set the miscellaneous information record for an image

Description

A CASA `image file` can accumulate miscellaneous information during its lifetime; it is stored in a record called the `miscinfo` record. For example, the FITS reader (fromfits) puts header keywords it doesn't otherwise use into the `miscinfo` record. The `miscinfo` record is not guaranteed to have any entries, so it's up to you to check for any fields that you require.
This function sets the `miscinfo` record of the `image file`. Note that this function *replaces* the record, it doesn't add to it, so if you want to augment the existing record, you should first capture it with the miscinfo function, add to the record, and then put it back. The FITS writer will attempt to write all the fields in the `miscinfo` record to the FITS file. It can do so for scalars and 1-dimensional arrays. Records will be omitted, and multi-dimensional arrays will be flattened into 1-dimensional arrays.

Arguments

Inputs		
info	Miscellaneous	REPLACEMENT header
	allowed:	record
	Default:	

Returns
bool

Example

```
"""
#
print "\t----\t setmiscinfo Ex 1 \t----"
ia.maketestimage('myfile',overwrite=true)
```

```
info = ia.miscinfo()            # capture the miscinfo record
info['extra'] = 'a test entry'  # add our information
ia.setmiscinfo(info)            # put it back into the image
ia.close()
#
"""
```

---

image.shape.html

image.shape - Function

1.1.1 Length of each axis in the image

Description

The shape of an image is a vector holding the length of each axis of the image. Although this information is also available in the summary function, it is so useful that it can be obtained directly. Both Float and Complex valued images are supported.

Arguments

Returns
intArray

Example

```
"""
#
print "\t----\t shape Ex 1 \t----"
ia.fromshape(shape=[10,20,30])
imshape = ia.shape()
print imshape
#[10L, 20L, 30L]
# npixels = imshape[0]*imshape[1]*...*imshape[n-1]
npixels=1
for i in range(len(imshape)):
  npixels=npixels*imshape[i]
ia.close()
#
"""
```

image.setrestoringbeam.html

image.setrestoringbeam - Function

1.1.1 Set the restoringbeam

Description

This function sets the restoring beam(s) for an image.
You may supply the beam in one of two ways.
First, you can use the argument `beam` which you must assign to a record containing fields 'major', 'minor' and 'positionangle'. Each of these fields contains a quantity. This record is in the same format as one returned by function restoringbeam. If `beam` is used, the arguments `major, minor, & pa` are ignored.
Second, you can use the arguments `major`, `minor` and `pa`. Only the ones that you assign are used. Each argument should be assigned either a quantity or a float (units are implicitly those of the current beam - or if none, arcsec for the axes and degrees for the position angle). These parameters are only used if `beam` is not specified.
An image must have exactly one of the following states:
1. An image can have a single "traditional" beam. In that case, the beam applies to every channel and polarization in the image.
2. If an image has more than one spectral channel or more than one polarization, it can have a set of beams. In this case, each channel and/or polarization will have its own beam.
3. An image can have neither a traditional beam nor a beam set.
It is never permissible for an image to have both a traditional (global) beam and a set of per-plane beams. Task and method behavior is undefined in that case and any resulting products are considered corrupt.
RULES FOR BEAM MODIFICATION
If an image has no beams, a traditional (global) beam can be added by setting both channel and polarization to negative values.
If an image has no beams, a set of per-plane beams can be added by setting either or both channel and/or polarization to a non-negative value. In this case, a number of per-plane beams are added consistent with the image and they are all set to be the same with parameters equal to those specified by either the beam or major/minor/pa parameters.
If an image has a traditional beam, it can be modified by setting both channel and polarization to negative values. If one or both is not set to a negative value, an exception is thrown, and nothing is modified.
If an image has a set of per plane beams, one at a time of these can be modified by setting the appropriate channel number and/or polarization

number. All the per-plane beams can be modified to the same values in one go by setting both channel and polarization to negative values. Also, in the case where an image has multiple channels, the beams associated with all channels for a given polarization can be modified to the same beam by setting polarization equal to the desired polarization plane number and by setting channel to a negative value. Similarly, in the case where an image has multiple polarizations, the beams associated with all polarizations for a given spectral channel can be modified to the same beam by setting channel equal to the desired spectral channel number and by setting polarization to a negative value.
A beam or set of beams can be copied from another image using the imagename parameter to specify that image's name. If both the current image and specified image have multiple beams, the current image shape must be consistent with the specified image beam set shape.
The traditional beam or a set of multiple beams can be deleted from an image by setting `delete=T`. If set to true, all other parameters are then ignored; all existing beams will be irrevocably deleted.

Arguments

Inputs	
major	Major axis FWHM, Quantity or float (e.g., 1arcsec). Default is unset. allowed: any Default: variant 1arcsec
minor	Minor axis FWHM, Quantity or float (e.g., 1arcsec). Default is unset. allowed: any Default: variant 1arcsec
pa	Position angle, Quantity or float (e.g., '5deg'). Default is unset. allowed: any Default: variant 0deg
beam	The complete restoring beam (output of restoringbeam()). Default is unset. allowed: record Default:
remove	Delete the restoring beam? allowed: bool Default: false
log	Write new beam values to the logger? allowed: bool Default: true
channel	Zero-based channel number for which to set a per plane beam. If the image has a traditional beam, set to less than zero. Default -1. allowed: int Default: -1
polarization	Zero-based polarization number for which to set a per plane beam. If the image has a traditional beam, set to less than zero. Default -1. allowed: int Default: -1
imagename	Copy the beam(s) from the specified image to this image. If multiple beams, the current image must be able to hold a beam set of the shape in the specified image. allowed: string Default:

Returns
bool

Example

```
"""

ia.maketestimage('hcn',overwrite=true)
rb = ia.restoringbeam()        # returns beam in record
print rb
#{'major': {'unit': 'arcsec', 'value': 53.500004857778549},
# 'minor': {'unit': 'arcsec', 'value': 34.199998900294304},
# 'positionangle': {'unit': 'deg', 'value': 6.0}}
rb['minor']['value'] = 12.5
# new beam specified in record
# NOTE This will not work for an image with multiple beams
ia.setrestoringbeam(beam=rb)
print ia.restoringbeam()
#{'major': {'unit': 'arcsec', 'value': 53.500004857778549},
# 'minor': {'unit': 'arcsec', 'value': 12.5},
# 'positionangle': {'unit': 'deg', 'value': 6.0}}

# beam specified using parameter
# NOTE This will only work for an image with a traditional beam
ia.setrestoringbeam(major='36arcsec')
print ia.restoringbeam()
#{'major': {'unit': 'arcsec', 'value': 36.0},
# 'minor': {'unit': 'arcsec', 'value': 12.5},
# 'positionangle': {'unit': 'deg', 'value': 6.0}}
ia.setrestoringbeam(remove=true)
print ia.restoringbeam()
#{}
ia.setrestoringbeam(major='53.5arcsec',minor='34.2arcsec',pa='6deg')
print ia.restoringbeam()
#{'major': {'unit': 'arcsec', 'value': 53.5},
# 'minor': {'unit': 'arcsec', 'value': 34.200000000000003},
# 'positionangle': {'unit': 'deg', 'value': 6.0}}
ia.close()

# Copy all beams from an image with multiple beams to another
# image with the same number of channels and polarizations

ia.open("multibeam.im")
ib = iatool()
ib.open("target.im")

# ensure target has no beam(s) at start, not always necessary
```

```
# but it doesn't hurt to do it.
ib.setrestoringbeam(remove=True)
# Now copy the beams. This only will work correctly if both images
# have the same number of channels and polarizations. nchan is set to
# the number of channels and npol is set to the number of polarizations
for c in range(nchan):
    for p in range(npol):
        beam = ia.restoringbeam(channel=c, polarization=p)
        ib.setrestoringbeam(beam=beam, channel=c, polarization=p)

ia.done()
ib.done()


"""
```

---

image.statistics.html

image.statistics - Function

1.1.1 Compute statistics from the image

Description

This function computes statistics from the pixel values in the image. You can then list them and retrieve them (into a record) for further analysis.
The chunk of the image over which you evaluate the statistics is specified by an array of axis numbers (argument `axes`). For example, consider a 3-dimensional image for which you specify `axes=[0,2]`. The statistics would be computed for each XZ (axes 0 and 2) plane in the image. You could then examine those statistics as a function of the Y (axis 1) axis. Or perhaps you set `axes=[2]`, whereupon you could examine the statistics for each Z (axis 2) profile as a function of X and Y location in the image.
Each statistic is stored in an array in one named field in the returned record. The shape of that array is that of the axes which you did **not** evaluate the statistics over. For example, in the second example above, we set `axes=[2]` and asked for statistics as a function of the remaining axes, in this case, the X and Y (axes 0 and 1) axes. The shape of each statistics array is then [nx,ny]. The names of the fields in this record are the same as the names of the statistics that you can plot:

- `npts` - the number of unmasked points used
- `sum` - the sum of the pixel values: $\sum I_i$
- `flux` - flux or flux density, see below for details
- `sumsq` - the sum of the squares of the pixel values: $\sum I_i^2$
- `mean` - the mean of pixel values: $\bar{I} = \sum I_i/n$
- `sigma` - the standard deviation about the mean: $\sigma^2 = (\sum I_i - \bar{I})^2/(n-1)$
- `rms` - the root mean square: $\sqrt{\sum I_i^2/n}$
- `min` - minimum pixel value
- `max` - the maximum pixel value
- `median` - the median pixel value (if `robust=T`)
- `medabsdevmed` - the median of the absolute deviations from the median (if `robust=T`)

- `quartile` - the inter-quartile range (if `robust=T`). Find the points which are 25% largest and 75% largest (the median is 50% largest).
- `q1` - The first quartile. Reported only if robust=T.
- `q3` - The third quartile. Reported only if robust=T.
- `blc` - the absolute pixel coordinate of the bottom left corner of the bounding box of the region of interest. If 'region' is unset, this will be the bottom left corner of the whole image.
- `blcf` - the formatted absolute world coordinate of the bottom left corner of the bounding box of the region of interest.
- `trc` - the absolute pixel coordinate of the top right corner of the bounding box of the region of interest.
- `trcf` - the formatted absolute world coordinate of the top right corner of the bounding box of the region of interest.
- `minpos` - absolute pixel coordinate of minimum pixel value
- `maxpos` - absolute pixel coordinate of maximum pixel value
- `minposf` - formatted string of the world coordinate of the minimum pixel value
- `maxposf` - formatted string of the world coordinate of the maximum pixel value

The last four fields only appear if you evaluate the statistics over all axes in the image. As an example, if the returned record is captured in `'mystats'`, then you could access the 'mean' field via `print mystats['mean']`.
If there are no good points (e.g. all pixels are masked bad in the region), then the length of these fields will be 0 (e.g. `len(mystats['npts'])==0`).
You have no control over which statistics are listed to the logger, you always get the same selection. You can choose to list the statistics or not (argument `list`).
As well as the simple (and faster to calculate) statistics like means and sums, you can also compute some robust (quantile-like) statistics. Currently these are the median, median absolute deviations from the median, the first and third quartiles, and the inner-quartile range. Because these are computationally expensive, they are only computed if robust=True.
Note that if the axes are set to all of the axes in the image (which is the default) there is just one value per statistic.
You have control over which pixels are included in the statistics computations via the `includepix` and `excludepix` arguments. These vectors specify a range of pixel values for which pixels are either included or excluded. They are mutually exclusive; you can specify one or the other, but not both. If you only

give one value for either of these, say `includepix=b`, then this is interpreted as `includepix=[-abs(b),abs(b)]`.
This function generates a 'storage' lattice, into which the statistics are written. It is only regenerated when necessary. For example, if you run the function twice with identical arguments, the statistics will be directly retrieved from the storage lattice the second time. However, you can force regeneration of the storage image if you set `force=T`. The storage medium is either in memory or on disk, depending upon its size. You can force it to disk if you set `disk=T`, otherwise it decides for itself.
ALGORITHMS
Several types of statistical algorithms are supported:
* classic: This is the familiar algorithm, in which all unmasked pixels, subject to any specified pixel ranges, are used. One may choose one of two methods, which vary only by performance, for computing classic statistics, via the clmethod parameter. The "tiled" method is the old method and is fastest in cases where there are a large number of individual sets of statistics to be computed and a small number of data points per set. This can occur when one sets the axes parameter, which causes several individual sets of statistics to be computed. The "framework" method uses the new statistics framework to compute statistics. This method is fastest in the regime where one has a small number of individual sets of statistics to calculate, and each set has a large number of points. For example, this method is fastest when computing statistics over an entire image in one go (no axes specified). A third option, "auto", chooses which method to use by predicting which be faster based on the number of pixels in the image and the choice of the axes parameter.
* fit-half: This algorithm calculates statistics on a dataset created from real and virtual pixel values. The real values are determined by the input parameters center and lside. The parameter center tells the algorithm where the center value of the combined real+virtual dataset should be. Options are the mean or the median of the input image's pixel values, or at zero. The lside parameter tells the algorithm on which side of this center the real pixel values are located. True indicates that the real pixel values to be used are ¡= center. False indicates the real pixel values to be used are ¿= center. The virtual part of the dataset is then created by reflecting all the real values through the center value, to create a perfectly symmetric dataset composed of a real and a virtual component. Statistics are then calculated on this resultant dataset. These two parameters are ignored if algorithm is not "fit-half". Because the maximum value is virtual if lside is True and the minimum value is virtual if lside is False, the value of the maximum position (if lside=True) or minimum position (if lside=False) is not reported in the returned record.
* hinges-fences: This algorithm calculates statistics by including data in a range between Q1 - f*D and Q3 + f*D, inclusive, where Q1 is the first quartile of the distribution of unmasked data, subject to any specified pixel ranges, Q3 is the third quartile, D = Q3 - Q1 (the inner quartile range), and f is the user-specified fence factor. Negative values of f indicate that the full distribution is to be used (ie, the classic algorithm is used). Sufficiently large

values of f will also be equivalent to using the classic algorithm. For f = 0, only data in the inner quartile range is used for computing statistics. The value of fence is silently ignored if algortihm is not "hinges-fences".
* chauvenet: The idea behind this algorithm is to eliminate outliers based on a maximum z-score value. A z-score is the number of standard deviations a point is from the mean of a distribution. This method thus is meant to be used for (nearly) normal distributions. In general, this is an iterative process, with successive iterations discarding additional outliers as the remaining points become closer to forming a normal distribution. Iterating stops when no additional points lie beyond the specified zscore value, or, if zscore is negative, when Chauvenet's criterion is met (see below). The parameter maxiter can be set to a non-negative value to prematurely abort this iterative process. When verbose=T, the "N iter" column in the table that is logged represents the number of iterations that were executed.
Chauvenet's criterion allows the target z-score to decrease as the number of points in the distribution decreases on subsequent iterations. Essentially, the criterion is that the probability of having one point in a normal distribution at a maximum z-score of z_max must be at least 0.5. z_max is therefore a function of (only) the number of points in the distrbution and is given by
npts = 0.5/erfc(z_max/sqrt(2))
where erfc() is the complementary error function. As iterating proceeds, the number of remaining points decreases as outliers are discarded, and so z_max likewise decreases. Convergence occurs when all remaining points fall within a z-score of z_max. Below is an illustrative table of z_max values and their corresponding npts values. For example, it is likely that there will be a 5-sigma "noise bump" in a perfectly noisy image with one million independent elements.
z_max npts 1.0 1 1.5 3 2.0 10 2.5 40 3.0 185 3.5 1,074 4.0 7,893 4.5 73,579 5.0 872,138 5.5 13,165,126 6.0 253,398,672 6.5 6,225,098,696 7.0 195,341,107,722
NOTES ON FLUX DENSITIES AND FLUXES
Fluxes and flux densities are not computed if any of the following conditions is met:
1. The image does not have a direction coordinate 2. The image does not have a intensity-like brightness unit. Examples of such units are Jy/beam (in which case the image must also have a beam) and K. 3. There are no direction axes in the cursor axes that are used. 4. If the (specified region of the) image has a non-degenerate spectral axis, and the image has a tablular spectral axis (axis with varying increments) 5. Any axis that is not a direction nor a spectral axis that is included in the cursor axes is not degenerate within in the specified region
Note that condition 4 may be removed in the future.
In cases where none of the above conditions is met, the flux density(ies) (intensities integrated over direction planes) will be computed if any of the following conditions are met:
1. The image has no spectral coordinate 2. The cursor axes do not include the spectral axis 3. The spectral axis in the chosen region is degenerate

In the case where there is a nondegenerate spectral axis that is included in the cursor axes, the flux (flux density integrated over spectral planes) will be computed. In this case, the spectral portion of the flux unit will be the velocity unit of the spectral coordinate if it has one (eg, if the brightness unit is Jy/beam and the velocity unit is km/s, the flux will have units of Jy.km/s). If not, the spectral portion of the flux unit will be the frequency unit of the spectral axis (eg, if the brightness unit is K and the frequency unit is Hz, the resulting flux unit will be K.arcsec2.Hz).
In both cases of flux density or flux being computed, the resulting numerical value is assigned to the "flux" key in the output dictionary.

Arguments

Inputs

axes	List of axes to evaluate statistics over. Default is all axes. allowed: intArray Default: -1
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
includepix	Range of pixel values to include. Vector of 1 or 2 doubles. Default is to include all pixels. allowed: doubleArray Default: -1
excludepix	Range of pixel values to exclude. Vector of 1 or 2 doubles. Default is exclude no pixels. allowed: doubleArray Default: -1
list	If True print bounding box and statistics to logger. allowed: bool Default: false
force	If T then force the stored statistical accumulations to be regenerated allowed: bool Default: false
disk	If T then force the storage image to disk allowed: bool Default: false
robust	If T then compute robust statistics as well allowed: bool Default: false
verbose	If T then log statistics allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false
logfile	Name of file to which to write statistics. allowed: string Default:
append	Append results to logfile? Logfile must be specified. Default is to append. False means overwrite existing file if it exists. allowed: bool Default: true
algorithm	Algorithm to use. Supported values are "chauvenet", "classic", "fit-half", and "hinges-fences". Minimum match is supported. allowed: string Default: classic
fence	Fence value for hinges-fences. A negative value means

Returns
record

Example

```
"""
#
print "\t----\t statistics Ex 1 \t----"
ia.maketestimage()
ia.statistics()
ia.close()
#

# evaluate statistics for each spectral plane in an ra x dec x frequency image
ia.fromshape("", [20,30,40])
# give pixels non-zero values
ia.addnoise()
# These are the display axes, the calculation of statistics occurs
# for each (hyper)plane along axes not listed in the axes parameter,
# in this case axis 2 (the frequency axis)
# display the rms for each frequency plane (your mileage will vary with
# the values).
stats = ia.statistics(axes=[0,1])
 stats["rms"]
  Out[10]:
array([ 0.99576014,  1.03813124,  0.97749186,  0.97587883,  1.04189885,
        1.03784776,  1.03371549,  1.03153074,  1.00841606,  0.947155  ,
        0.97335404,  0.94389403,  1.0010221 ,  0.97151822,  1.03942156,
        1.01158476,  0.96957082,  1.04212773,  1.00589049,  0.98696715,
        1.00451481,  1.02307892,  1.03102005,  0.97334671,  0.95209879,
        1.02088714,  0.96999902,  0.98661619,  1.01039267,  0.96842754,
        0.99464947,  1.01536798,  1.02466023,  0.96956468,  0.98090756,
        0.9835844 ,  0.95698935,  1.05487967,  0.99846411,  0.99634868])


"""

In this example, we ask to see statistics evaluated over the
entire image.
```

Example

```
"""
#
print "\t----\t statistics Ex 2 \t----"
ia.maketestimage()
stats = ia.statistics(axes=[1],plotstats=["sigma","rms"],
                      includepix=[0,100],list=F)
#
"""
```

In this example, let us assume the image has 2 dimensions. We want the standard deviation about the mean and the rms of Y (axes 1) for pixels with values in the range 0 to 100 as a function of the X-axis location. The statistics are not listed to the logger but are saved in the record {\stfaf 'stats'}.

---

image.twopointcorrelation.html

image.twopointcorrelation - Function

1.1.1 Compute two point correlation function from the image

Description

This function computes two-point auto-correlation functions from an image. By default, the auto-correlation function is computed for the Sky axes. If there is no sky in the image, then the first two axes are used. Otherwise you can specify which axes the auto-correlation function lags are computed over with the `axes` argument (must be of length 2).
Presently, only the Structure Function is implemented. This is defined as :

$$S(lx, ly) =< (data(i, j) - data(i + lx, j + ly))^2 >$$

where lx, ly are integer lags in the x (0-axis) and y (1-axis) directions. The ensemble average is over all the values at the same lag pair. This process is extremely compute intensive and so you may have to be patient.
In an auto-correlation function image there are some symmetries. The first and third quadrants are symmetric, and the second and fourth are symmetric. So in principle, all the information is in the top or bottom half of the image. We just write it all out to look nice. The long lags don't have a lot of contributing values of course.

Arguments

Inputs	
outfile	Output image file name. Default is unset. allowed: string Default:
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
axes	The pixel axes to compute structure function over. The default is sky or first two axes. allowed: intArray Default: -1
method	The method of computation. String from 'structurefunction'. allowed: string Default: structurefunction
overwrite	Overwrite (unprompted) pre-existing output file? allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false

Returns
bool

Example

```
"""
#
print "\t----\t twopointcorrelation Ex 1 \t----"
ia.maketestimage();        # Output image is virtual
ia.twopointcorrelation()   # Output image is virtual
#
```

"""

---

image.subimage.html

image.subimage - Function

1.1.1 Create a (sub)image from a region of the image

Description

This function copies all or part of the image to another on-the-fly Image tool. Both float and complex valued images are supported.
If `outfile` is given, the subimage is written to the specified disk file. If `outfile` is unset, the returned Image `tool` actually references the input image file (i.e. that associated with the Image `tool` to which you are applying this function). So if you deleted the input image disk file, it would render this `tool` useless. When you destroy this `tool` (with the done function) the reference connection is broken.
Sometimes it is useful to drop axes of length one (degenerate axes). Use the `dropdeg` argument if you want to do this. Further control is provided via the keepaxes parameter. If dropdeg=True, you may specify a list of degenerate axes to keep in the keep axes parameter. This allows you to drop only a subset of degenerate axes. This parameter is ignored if dropdeg=False. If dropdeg=True, all degenerate axes are dropped if keepaxes is set to an empty list (this is the default behavior). Nondegenerate axes are implicitly kept, even if they are included in the keepaxes list.
The output mask is the combination (logical OR) of the default input `pixel mask` (if any) and the OTF mask. Any other input `pixel masks` will not be copied. Use function maskhandler if you need to copy other masks too.
If the mask has fewer dimensions than the image and if the shape of the dimensions the mask and image have in common are the same, the mask will automatically have the missing dimensions added so it conforms to the image. If stretch is true and if the number of mask dimensions is less than or equal to the number of image dimensions and some axes in the mask are degenerate while the corresponding axes in the image are not, the mask will be stetched in the degenerate dimensions. For example, if the input image has shape [100, 200, 10] and the input mask has shape [100, 200, 1] and stretch is true, the mask will be stretched along the third dimension to shape [100, 200, 10]. However if the mask is shape [100, 200, 2], stretching is not possible and an error will result.

Arguments

Inputs	
outfile	Output image file name. Default is unset. allowed: string Default:
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
dropdeg	Drop degenerate axes allowed: bool Default: false
overwrite	Overwrite (unprompted) pre-existing output file? allowed: bool Default: false
list	List informative messages to the logger allowed: bool Default: true
stretch	Stretch the mask if necessary and possible? allowed: bool Default: false
wantreturn	Return an image analysis tool attached to the created subimage allowed: bool Default: true
keepaxes	If dropdeg=True, these are the degenerate axes to keep. Nondegenerate axes are implicitly always kept. allowed: intArray Default:

Returns
image

Example

"""

```
#
print "\t----\t subimage Ex 1 \t----"
ia.maketestimage('myfile',overwrite=true)
im2 = ia.subimage()                # a complete copy
r1 = rg.box([10,10],[30,40],[5,5]) # A strided pixel box region
im3 = ia.subimage(outfile='/tmp/foo', region=r1, overwrite=true)
                                   # Explicitly named subimage
im2.done()
im3.done()
ia.close()
#
"""
```

---

image.summary.html

image.summary - Function

1.1.1 Summarize basic information about the image

Description

This function summarizes miscellaneous information such as shape, Coordinate System, restoring beams, and masks.
If called without any arguments, this function displays a summary of the image header to the logger; where appropriate, values will be formatted nicely (e.g. HH:MM:SS.SS for the reference value of RA axes).
For spectral axes, the information is listed as a velocity as well as a frequency. The argument `doppler` allows you to specify what velocity doppler convention it is listed in. You can choose from `radio, optical` and `true`. Alternative names are `z` for `optical`, and `beta` or `relativistic` for `true`. The default is `radio`. The definitions are

- radio: $1 - F$
- optical: $-1 + 1/F$
- true: $(1 - F^2)/(1 + F^2)$

where $F = \nu/\nu_0$ and ν_0 is the rest frequency. If the rest frequency has not been set in your image, you can set it via a Coordsys `tool` with the function setrestfrequency.
If the output of summary is saved to a variable, then the `header` field (for instance, `mysummary['header']`) has the following fields filled in:

ndim Dimension of the image.

shape Length of each axis in the image.

tileshape Shape of the chunk which is most efficient for I/O.

axisnames Name of each axis.

refpix Reference pixel for each axis (0-relative)

refval Reference value for each axis.

incr Increment for each axis.

axisunits Unit name for each axis.

unit Brightness units for the pixels.

hasmask True if the image has a mask.

defaultmask The name of the mask which is applied by default.

masks The names of all the masks stored in this image.

restoringbeam The restoring beam(s) if present.

imagetype The image type.

For an image with multiple beams, the restoringbeam field is a dictionary of dictionaries with keys of names "*" followed by the channel number, if the image has a spectral coordinate, or the polarization number if it does not. That is, the keys have names "*0", "*1", "*2", etc. If the image has both a spectral and a polarization coordinate, each of these dictionaries is a dictionary with keys of the same form which range from 0 to the number of polarizations minus 1; "*0", "*1", ... The dictionaries pointed to by the channel and/or polarization number contain information for the beam at that position.
If you set `list=F`, then the summary will not be written to the logger. The return value of the function, in the `header` field is a vector string containing the formatted output that would normally have gone to the logger.
If verbose is True and the image contains multiple beams, the formatted output, whether it is written to the logger or placed in the output record, will have information on every beam in the dataset. If verbose=False and the image has multiple beams, only a summary of beams for each polarization is listed. In this case, the beams with the maximum area, the minimum area, and the median area for each polarization are listed. However, all the beams can still be found in the restoringbeam field of the returned dictionary. If the image does not have multiple beams, verbose is not used.

Arguments

Inputs	
doppler	If there is a spectral axis, list velocity too, with this doppler definition
	allowed: string
	Default: RADIO
list	List the summary to the logger
	allowed: bool
	Default: true
pixelorder	List axis descriptors in pixel or world axis order
	allowed: bool
	Default: true
verbose	Give a full listing of beams or just a short summary? Only used when the image has multiple beams.
	allowed: bool
	Default: false

Returns
record

Example

```
"""
#
print "\t----\t summary Ex 1 \t----"
ia.maketestimage('myim1', overwrite=true)
ia.summary()                  # summarize to logging only
s = ia.summary(list=F)        # store header in record
if s['header']['ndim'] == 2: # program using header values
  print s['header']['axisnames']
ia.close()
#
"""
```

image.tofits.html

image.tofits - Function

1.1.1 Convert the image to a FITS file

Description

This function converts the image into a FITS file.
If the image has a rest frequency associated with it, it will always write velocity information into the FITS file. By default the frequency information will be primary as it is the internal native format. If you select `velocity=T` then by default the velocity is written in the optical convention, but if `optical=F` it will use the radio convention instead. Alternatively, if you use `velocity=F` and `wavelength=T`, the spectral axis will be written in wavelength.
The FITS definition demands equal increment pixels. Therefore, if you write wavelength or optical velocity information as primary, the increment is computed at the spectral reference pixel. If the bandwidth is large, this may incur non-negligible coordinate calculation errors far from the reference pixel if the spectral bins are not originally equidistant in wavelength. Images generated by the CASA clean task have spectral axes which are always equidistant in frequency.
By default the image is written as a floating point FITS file (`bitpix= -32`). Under rare circumstances you might want to save space and write it as scaled 16 bit integers (`bitpix = 16`). You can have `tofits` calculate the scaling factors by using the default `minpix` and `maxpix`. If you set `minpix` and `maxpix`, values outside of that range will be truncated. This can be useful if all of the FITS images dynamic range is being used by a few high or low values and you are not interested in preserving those values exactly. Besides the factor of two space savings you get by using 16 instead of 32 bits, integer images usually also compress well (for example, with the standard GNU software facility `gzip`).
If the specified `region-of-interest` extends beyond the image, it is truncated.
The output mask is the combination (logical OR) of the default input `pixel mask` (if any) and the OTF mask.
Sometimes it is useful to drop axes of length one (degenerate axes) because not all FITS readers can handle them. Use the `dropdeg` argument if you want to do this. If you want to specifically only drop a degenerate Stokes axis, use the `dropstokes` argument.
If you want to place degenerate axes last in the FITS header, use the `deglast` argument. If you want to make sure that the Stokes axis is placed last in the FITS header, use the `stokeslast` argument.

Arguments

Inputs	
outfile	FITS file name. Default is input name + '.fits' allowed: string Default:
velocity	prefer velocity (rather than frequency) as primary spectral axis? allowed: bool Default: false
optical	use the optical (rather than radio) velocity convention? allowed: bool Default: true
bitpix	Bits per pixel, -32 (floating point) or 16 (integer) allowed: int Default: -32
minpix	Minimum pixel value for BITPIX=16, Default is to autoscale if minpix > maxpix. allowed: double Default: 1
maxpix	Maximum pixel value for BITPIX=16, Default is to autoscale if maxpix < minpix. allowed: double Default: -1
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
overwrite	Overwrite (unprompted) pre-existing output file? allowed: bool Default: false
dropdeg	Drop degenerate axes? allowed: bool Default: false
deglast	Put degenerate axes last in header? allowed: bool Default: false
dropstokes	Drop Stokes axis? allowed: bool Default: false
stokeslast	Put Stokes axis last in header? allowed: bool Default: true
wavelength	Write spectral axis in units of wavelength (instead of velocity or frequency)? allowed: bool Default: false
airwavelength	When wirting the spectral axis in units of wavelength, use air wavelength instead of vacuum wavelength? allowed: bool Default: false
async	Run asynchronously? allowed: bool

Returns
bool

Example

```
"""
#
print "\t----\t tofits Ex 1 \t----"
ia.maketestimage()
ok = ia.tofits('MYFILE.FITS',overwrite=true)
                      # write FITS image file
ok = ia.tofits('MYFILE2.FITS', bitpix=16, overwrite=true)
                      # Write as scaled 16 bit integers
ia.close()
#
"""
```

---

image.toASCII.html

image.toASCII - Function

1.1.1 Convert the image to an ASCII file

Description

This function converts the image into an ascii file. The format is one image row per line (see fromascii).
The output mask is the combination (logical OR) of the default input `pixel mask` (if any) and the OTF mask. Because the mask is not transferred to the ascii file, you must specify what data value to use if a pixel is masked. By default, the underlying data value in the image is used. But this could be anything (and often it's a NaN), so you could set, say, `maskvalue=-10000` as a magic value.

Arguments

Inputs	
outfile	ASCII file name. Default is input name + '.ascii'. allowed: string Default:
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
mask	Mask to use. See help par.mask. Default is none. allowed: any Default: variant
sep	Separator of data in ascii file. Default is space character. allowed: string Default: :
format	Format of data in ascii file allowed: string Default:
maskvalue	Value to replace masked pixels by, -999 is no change. allowed: double Default: -999
overwrite	Overwrite (unprompted) pre-existing output file? allowed: bool Default: false
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false

Returns
bool

Example

```
"""
#
print "\t----\t toASCII Ex 1 \t----"
ia.maketestimage()
```

```
ok = ia.toASCII('myfile.ascii', overwrite=true)
ia.close()
ia.fromascii('image.im','myfile.ascii', shape=[113,76], overwrite=true)
ia.toASCII('myfile2.ascii',overwrite=true)
#!diff myfile.ascii myfile2.ascii
#
"""
```

---

image.torecord.html

image.torecord - Function

1.1.1 Return a record containg the image associated with this tool

Description

You can convert an associated image to a record for manipulation or passing it to inputs of other function of other tools

Arguments

Returns
record

Example

```
"""
#
print "\t----\t torecord Ex 1 \t----"
ia.maketestimage('image.large', overwrite=true)
rec=ia.torecord()
ia.close()


"""
```

image.type.html

image.type - Function

1.1.1 Return the type of this tool

Description

This function returns the string ‘image’. It can be used in a script to make sure this variable is an Image `tool`.

Arguments

Returns
string

---

image.topixel.html

image.topixel - Function

1.1.1 Convert from world to pixel coordinate

Description

This function converts from absolute world to pixel coordinate (0-rel). The world coordinate can be provided in many formats (numeric, string, quantum etc.) via the argument `value`. These match the output formats of function toworld.
This function is just a wrapper for the Coordsys `tool` function topixel so see the documentation there for a description and more examples.

Arguments

Inputs		
value	Absolute world coordinate, Numeric vector, vector of strings representing quantities, or record of format analogous to that produced by ia.toworld(). Default is reference value.	
	allowed:	any
	Default:	variant

Returns
record

Example

```
"""
#
print "\t----\t topixel Ex 1 \t----"
ia.maketestimage();
w = ia.toworld([10,10], 'n')        # Numeric vector
ia.topixel(w)
#{'ar_type': 'absolute',
```

```
# 'numeric': array([10., 10.]), 'pw_type': 'pixel'}
w = ia.toworld([10,10], 'm')        # Record of measures
ia.topixel(w)
#{'ar_type': 'absolute',
# 'numeric': array([10., 10.]), 'pw_type': 'pixel'}
ia.close()
#
"""

Convert a pixel coordinate to world as floats and then
back to pixel.  Do the same with the world coordinate
formatted as measures instead.
```

---

image.toworld.html

image.toworld - Function

1.1.1 Convert from pixel to world coordinate

Description

This function converts between absolute pixel coordinate (0-rel) and world (physical coordinate).
This function is just a wrapper for the Coordsys `tool` function toworld so see the documentation there for a description of the arguments and more examples.

Arguments

Inputs	
value	Absolute pixel coordinate. Numeric vector is allowed. Default is reference pixel. allowed: any Default: variant
format	What type of formatting? String from combination of 'n' (numeric), 'q' (quantity), 'm' (measure), 's' (string). allowed: string Default: n
dovelocity	Compute corresponding spectral velocities? allowed: bool Default: True

Returns
record

Example

```
"""
#
```

```
print "\t----\t toworld Ex 1 \t----"
ia.maketestimage('hcn',overwrite=true)
w = ia.toworld([10,10], 'n')
print w
#{'numeric': array([ 0.00174533, -0.0015708 ])}
w = ia.toworld([10,10], 'nmq')
print w
#{'measure': {'direction': {'m0': {'unit': 'rad',
#                                   'value': 0.0017453323593185704},
#                           'm1': {'unit': 'rad',
#                                  'value': -0.0015707969259645381},
#                           'refer': 'J2000',
#                           'type': 'direction'}},
# 'numeric': array([ 0.00174533, -0.0015708 ]),
# 'quantity': {'*1': {'unit': 'rad', 'value': 0.0017453323593185704},
#              '*2': {'unit': 'rad', 'value': -0.0015707969259645381}}}
ia.close()
#
"""

Convert to a vector of floats and then to a record
holding a vector of floats, a vector of quantities
and a subrecord of measures.
```

---

image.unlock.html

image.unlock - Function

1.1.1 Release any lock on the image

Description

This function releases any lock set on the `image file` (and also flushes any outstanding I/O to disk). It is not of general user interest. It can be useful in scripts when a file is being shared between more than one process. See also functions lock and haslock.

Arguments

Returns
bool

Example

```
"""
#
print "\t----\t unlock Ex 1 \t----"
ia.fromarray('xx', ia.makearray(0,[10,20]), overwrite=true)
ia.unlock()
ia.close()
#
"""


This releases the write lock on the \imagefile.  Now some
other process can gain immediate access to the \imagefile.
```

image.newimagefromarray.html

image.newimagefromarray - Function

1.1.1 Construct a CASA image from an array

Description

This function converts an array of any size into a CASA `image file`.
If `outfile` is given, the image is written to the specified disk file. If `outfile` is unset, the on-the-fly Image `tool` returned by this function is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the on-the-fly Image `tool` (with the done function) this temporary image is deleted.
At present, no matter what type the `pixels` array is, a real-valued image will be written (the input pixels will be converted to Float). In the future, Complex images will be supported.
The Coordinate System, provided as a record describing a Coordsys `tool` (via coordsys.torecord(), for instance) is optional. If you provide it, it must be dimensionally consistent with the pixels array you give (see also coordsys).
If you don't provide the Coordinate System (unset), a default Coordinate System is made for you. If `linear=F` (the default) then it is a standard RA/DEC/Stokes/Spectral Coordinate System depending exactly upon the shape of the `pixels` array (Stokes axis must be no longer than 4 pixels and you may find the spectral axis coming out before the Stokes axis if say, `shape=[64,64,32,4]`). Extra dimensions are given linear coordinates. If `linear=T` then you get a linear Coordinate System.

Arguments

Inputs		
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	
pixels	A numeric array is required.	
	allowed:	any
	Default:	variant
csys	Coordinate System. Default is unset.	
	allowed:	record
	Default:	
linear	Make a linear Coordinate System if csys not given	
	allowed:	bool
	Default:	false
overwrite	Overwrite (unprompted) pre-existing output file?	
	allowed:	bool
	Default:	false
log	Write image creation messages to logger	
	allowed:	bool
	Default:	true

Returns
image

Example

```
"""
#
print "\t----\t newimagefromarray Ex 1 \t----"
im1=ia.newimagefromarray(outfile='test.data',
                         pixels=ia.makearray(0, [64, 64, 4, 128]),
                         overwrite=true)
cs1 = im1.coordsys(axes=[0,1])
im1.done()
im2 = ia.newimagefromarray(pixels=ia.makearray(1.0, [32, 64]),
                           csys=cs1.torecord())
cs1.done()
im2.done()
#
"""
```

The first example creates a zero-filled \imagefile\ named {\sff test.data} which is of shape [64,64,4,128]. If you examine the header with {\stff ia.summary()} you will see the default RA/DEC/Stokes/Frequency coordinate information. In the second example, a Coordinate System describing the first two axes of the image {\sff test.data} is created and used to create a 2D image temporary image.

---

image.newimagefromfits.html

image.newimagefromfits - Function

1.1.1 Construct a CASA image by conversion from a FITS image file

Description

This function is used to convert a FITS disk image file (Float, Double, Short, Long are supported) to an CASA `image file`. If `outfile` is given, the image is written to the specified disk file. If `outfile` is unset, the on-the-fly Image `tool` returned by this function is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the on-the-fly Image `tool` (with the done function) this temporary image is deleted.
This function reads from the FITS primary array (when the image is at the beginning of the FITS file; `whichhdu=0`), or an image extension (when the image is elsewhere in the FITS file, `whichhdu` $>$ `0`).
By default, any blanked pixels will be converted to a mask value which is false, and a pixel value that is NaN. If you set `zeroblanks=T` then the pixel value will be zero rather than NaN. The mask will still be set to false. See the function replacemaskedpixels if you need to replace masked pixel values after you have created the image.

Arguments

Inputs		
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	
infile	Input FITS disk file name. Required.	
	allowed:	string
	Default:	
whichrep	If this FITS file contains multiple coordinate representations, which one should we read	
	allowed:	int
	Default:	0
whichhdu	If this FITS file contains multiple images, which one should we read (0-based).	
	allowed:	int
	Default:	0
zeroblanks	If there are blanked pixels, set them to zero instead of NaN	
	allowed:	bool
	Default:	false
overwrite	Overwrite (unprompted) pre-existing output file?	
	allowed:	bool
	Default:	false

Returns
image

Example

```
"""
#
print "\t----\t newimagefromfits Ex 1 \t----"
# Assume we can find test fits file using
# CASAPATH environment variable
pathname=os.environ.get("CASAPATH")
pathname=pathname.split()[0]
datapath=pathname+'/data/demo/Images/imagetestimage.fits'
im1=ia.newimagefromfits('./myimage', datapath, overwrite=true)
print im1.summary()
print im1.miscinfo()
print 'fields=', im1.miscinfo().keys()
```

```
im1.done()
#
"""
```

The FITS image is converted to a \casa\ \imagefile\ and access is provided via the \imagetool\ called {\stf im1}. Any FITS header keywords which were not recognized or used are put in the miscellaneous information bucket accessible with the miscinfo function. In the example we list the names of the fields in this record.

---

image.newimagefromimage.html

image.newimagefromimage - Function

1.1.1 Construct an on-the-fly image tool from a region of a CASA image file

Description

This function applies a `region-of-interest` to a disk `image file`, creates a new `image file` containing the (sub)image, and associates a new `image tool` with it.
The input disk image file may be in native CASA, FITS(Float, Double, Short, Long are supported), or Miriad format. Look here for more information on foreign images.
If `outfile` is given, the (sub)image is written to the specified disk file.
If `outfile` is unset, the Image `tool` actually references the input image file. So if you deleted the input image disk file, it would render this `tool` useless. When you destroy this on-the-fly `tool` (with the done function) the reference connection is broken.
Sometimes it is useful to drop axes of length one (degenerate axes). Use the `dropdeg` argument if you want to do this.
The output mask is the combination (logical OR) of the default input `pixel mask` (if any) and the OTF mask. Any other input `pixel masks` will not be copied. Use function maskhandler if you need to copy other masks too.
See also the subimage function.

Arguments

Inputs		
infile	Input image file name. Required.	
	allowed:	string
	Default:	
outfile	Output sub-image file name. Default is unset.	
	allowed:	string
	Default:	
region	Region selection. See "help par.region" for details. Default is to use the full image.	
	allowed:	any
	Default:	variant
mask	Mask to use. See help par.mask. Default is none.	
	allowed:	any
	Default:	variant
dropdeg	Drop degenerate axes	
	allowed:	bool
	Default:	false
overwrite	Overwrite (unprompted) pre-existing output file?	
	allowed:	bool
	Default:	false

Returns
image

Example

```
"""
#
print "\t----\t newimagefromimage Ex 1 \t----"
ia.maketestimage('test1',overwrite=true)
ia.maketestimage('test2',overwrite=true)
print ia.name()
#[...]test2
im1=ia.newimagefromimage('test1')
print im1.name()
#[...]test1
print im1.summary()
im2=ia.newimagefromimage('test2')
```

```
print im2.name()
#[...]test2
print im1.name()
#[...]test1
ia.close()
im1.done()
im2.done()
#
"""
```

---

image.newimagefromshape.html

image.newimagefromshape - Function

1.1.1 Construct an empty CASA image from a shape

Description

This function creates a CASA `image file` with the specified shape. All the pixel values in the image are set to 0. Both float valued and complex valued images are supported; the data type of the image is specified via the type parameter.
If `outfile` is given, the image is written to the specified disk file. If `outfile` is unset, the on-the-fly Image `tool` returned by this function is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the on-the-fly Image `tool` (with the done function) this temporary image is deleted.
The Coordinate System, provided as a record describing a Coordsys `tool` (created via coordsys.torecord(), for instance), is optional. If you provide it, it must be dimensionally consistent with the pixels array you give (see also coordsys).
If you don't provide the Coordinate System, a default Coordinate System is made for you. If `linear=F` (the default) then it is a standard RA/DEC/Stokes/Spectral Coordinate System depending exactly upon the shape (Stokes axis must be no longer than 4 pixels and you may find the spectral axis coming out before the Stokes axis if say, `shape=[64,64,32,4]`). Extra dimensions are given linear coordinates. If `linear=T` then you get a linear Coordinate System.

Arguments

Inputs		
outfile	Name of output image file. Default is unset.	
	allowed:	string
	Default:	
shape	Shape of image. Required.	
	allowed:	intArray
	Default:	0
csys	Record describing Coordinate System. Default is unset.	
	allowed:	record
	Default:	
linear	Make a linear Coordinate System if csys not given?	
	allowed:	bool
	Default:	false
overwrite	Overwrite (unprompted) pre-existing output file?	
	allowed:	bool
	Default:	false
log	Write image creation messages to logger	
	allowed:	bool
	Default:	true
type	Type of image. 'f' means Float, 'c' means complex.	
	allowed:	string
	Default:	f

Returns
image

Example

```
"""
#
print "\t----\t newimagefromshape Ex 1 \t----"
im1=ia.newimagefromshape('test2.data', [64,64,128], overwrite=true)
cs1 = im1.coordsys(axes=[0,2])
im1.done()
im2 = ia.newimagefromshape(shape=[10, 20], csys=cs1.torecord())
cs1.done()
im2.done()
#
"""
```

The first example creates a zero-filled \imagefile\ named {\sff test.data} of shape [64,64,128]. If you examine the header with {\stff ia.summary()} you will see the RA/DEC/Spectral coordinate information. In the second example, a Coordinate System describing the first and third axes of the image {\sff test2.data} is created and used to create a 2D temporary image.

---

image.pbcor.html

image.pbcor - Function

1.1.1 Construct a primary beam corrected image from an image and a primary beam

Description

Correct an image for primary beam attenuation using an image of the primary beam pattern. The primary beam pattern can be provided as an image, in which case 1. it must have the same shape as the input image and its coordinate system must be the same, or 2. it must be a 2-D image in which case its coordinate system must consist of a (2-D) direction coordinate which is the same as the direction coordinate in the input image and its direction plane must be the same shape as that of the input image. Alternatively, pbimage can be an array of pixel values in which case the same dimensionality and shape constraints apply. An image tool referencing the corrected image is returned. The corrected image will also be written to disk if outfile is not empty (and overwrite=True if outfile already exists). One can choose between dividing the image by the primary beam pattern (mode="divide") or multiplying the image by the primary beam pattern (mode="multiply"). One can also choose to specify a cutoff limit for the primary beam pattern. For mode="divide", for all pixels below this cutoff in the primary beam pattern, the output image will be masked. In the case of mode="multiply", all pixels in the output will be masked corresponding to pixels with values greater than the cutoff in the primary beam pattern. A negative value for cutoff means that no cutoff will be applied, which is the default.

Arguments

Inputs	
pbimage	Name of the primary beam image which must exist or array of values for the pb response. Default "" allowed: any Default: variant
outfile	Output image name. If empty, no image is written. Default "" allowed: string Default:
overwrite	Overwrite the output if it exists? Default False allowed: bool Default: false
box	Rectangular region(s) to select in direction plane. See "help par.box" for details. Default is to use the entire direction plane. allowed: string Default:
region	Region selection. See "help par.region" for details. Default is to use the full image. allowed: any Default: variant
chans	Channels to use. See "help par.chans" for details. Default is to use all channels. allowed: string Default:
stokes	Stokes planes to use. See "help par.stokes" for details. Default is to use all stokes planes. allowed: string Default:
mask	Mask to use. See help par.mask. Default is none. allowed: string Default: variant
mode	Divide or multiply the image by the primary beam image. Minimal match supported. Default "divide" allowed: string Default: divide
cutoff	PB cutoff. If mode is "d", all values less than this will be masked. If "m", all values greater will be masked. Less than 0, no cutoff. Default no cutoff allowed: float Default: -1.0
stretch	Stretch the mask if necessary and possible? See help par.stretch. Default False allowed: bool Default: false

Returns
image

Example

```
ia.open("attenuated.im")
ia.pbcor(pbimage="mypb.im", outname="pbcorred.im", mode="divide", cutoff=0.1)
```

---

image.pv.html

image.pv - Function

1.1.1 Construct a position-velocity image between two points in the direction plane.

Description

Create a position-velocity image by specifying either two points between which a slice is taken in the direction coordinate or a center, position angle, and length describing the slice. The spectral extent of the resulting image will be that provided by the region specification or the entire spectral range of the input image if no region is specified. One may not specify a region in direction space; that is accomplished by specifying the start and end points or the center, length, and position angle of the slice. The parameters start and end may be specified as two element arrays of numerical values, in which case these values will be interpreted as pixel locations in the input image. Alternatively, they may be expressed as arrays of two strings each representing the direction. These strings can either represent quantities (eg [”40.5deg”, ”0.5rad”) or be sexigesimal format (eg [”14h20m20.5s”,”-30d45m25.4s”], [”14:20:20.5s”,”-30.45.25.4”]). In addition, they may be expressed as a single string containing the longitude and latitude values and optionally a reference frame value, eg ”J2000 14:20:20.5s -30.45.25.4”. The center parameter is specified in the same way. The length parameter may be specified as a single numerical value, in which case it is interpreted as the length in pixels, or a valid quantity, in which case it must have units conformant with the direction axes units. The pa (position angle) parameter must be specified as a valid quantity with angular units. The position angle is interpreted in the usual astronomical sense; ie measured from north through east. Either start/end or center/pa/length must be specified; if a parameter from one of these sets is specified, a parameter from the other set may not be specified. In either case, the end points of the segment must fail within the input image, and they both must be at least 2 pixels from the edge of the input image to facilite rotation (see below).
One may specify a width, which is the number of pixels centered along and perpendicular to the direction slice that are used for averaging along the slice. The width may be specified as an integer, in which case it must be positive and odd. Alternatively, it may be specified as a valid quantity string (eg, ”4arcsec”) or quantity record (eg qa.quantity(”4arcsec”). In this case, units must be conformant to the direction axes units (usually angular units) and the specified quantity will be rounded up, if necessary, to the next highest equivalent odd integer number of pixels. The default value of 1 represents no

averaging. A value of 3 means average one pixel on each side of the slice and the pixel on the slice. Note that this width is applied to pixels in the image after it has been rotated (see below for a description of the algorithm used). The end points of the specified segment must fail within the input image, and they both must be at least 2 pixels from the edge of the input image to facilite rotation (see below).
One may specify the unit for the angular offset axis.
A true value for the wantreturn parameter indicates that an image analysis tool attached to the created image should be returned. Nothing is returned if wantreturn is false, but then outfile should be specified (unless perhaps you are debugging).
Internally, the image is first rotated, padding first if necessary to include relevant pixels that would otherwise be excluded by the rotation operation, so that the slice is horizontal, with the starting pixel left of the ending pixel. Then, the pixels within the specified width of the slice are averaged and the resulting image is written and/or returned. The output image has a linear coordinate in place of the direction coordinate of the input image, and the corresponding axis represents angular offset with the center pixel having a value of 0.
The equivalent coordinate system, with a (usually) rotated direction coordinate (eg, RA and Dec) is written to the output image as a table record. It can be retrieved using the table tool as shown in the example below.

Arguments

Inputs

outfile	Output image name. If empty, no image is written. Default "" allowed: string Default:
start	The starting point in the direction plane (array of two values). If specified, end must also be specified and none of center, pa, nor length may be specified. allowed: any Default: variant
end	The ending point in the direction plane (array of two values). If specified, start must also be specified and none of center, pa, nor length may be specified. allowed: any Default: variant
center	The center point in the direction plane (array of two values). If specified, length and pa must also be specified and neither of start nor end may be specified. allowed: any Default: variant
length	The length of the segment in the direction plane. If specified, center and pa must also be specified and neither of start nor end may be specified. allowed: any Default: variant
pa	The position angle of the segment in the direction plane, measured from north through east. If specified, center and length must also be specified and neither of start nor end may be specified. allowed: any Default: variant
width	Width in pixels for averaging pixels perpendicular to the slice. Must be an odd integer >= 1 (1 means only use the pixels along the slice), or a quantity which will be rounded up if necessary so it corresponds to the next highest odd number of pixels. allowed: any Default: variant 1
unit	Unit for the offset axis in the resulting image. Must be a unit of angular measure. allowed: string Default: arcsec
overwrite	Overwrite the output if it exists? allowed: bool Default: false
region	Region selection. See "help par.region" for details. Default is to use the full image. No selection is permitted in the direction plane. allowed: any Default: variant

Returns
image

Example

```
ia.open("my_spectral_cube.im")
# create a pv image with the position axis running from ra, dec pixel positions of
# in the input image
mypv = ia.pv(outfile="pv.im", start=[45,50], end=[100,120], wantreturn=true)
ia.done()
# analyze the pv image, such as get statistics
pvstats = mypv.statistics()
# when done, close the tool to release system resources
mypv.done()

# get the alternate coordinate system information
tb.open("pv.im")
alternate_csys_record = tb.getkeyword("misc")["secondary_coordinates"]
tb.done()
```

---

image.makearray.html

image.makearray - Function

1.1.1 Construct an initialized multi-dimensional array.

Description

This function takes two arguments. The first argument is the initial value for the new array. The second is a vector giving the lengths of the dimensions of the array.

Arguments

Inputs		
v	Value with which to initial array elements	
	allowed:	double
	Default:	0.0
shape	Vector containing array dimensions.	
	allowed:	intArray
	Default:	0

Returns

anyvariant

Example

```
A three dimensional array that is initialized to all zeros. Each of
the three dimensions of the cube has a length of four.

"""
#
print "\t----\t makearray Ex 1 \t----"
cube = ia.makearray(0,[4,4,4])
#
"""
```

image.isconform.html

image.isconform - Function

1.1.1 Returns true of the shape, coordinate system, and axes order of the specified image matches this image.

Description

Returns True if the shape, coordinate system, and axes order of the specified image matches the current image.

Arguments

Inputs		
other	name of image to test	
	allowed:	string
	Default:	

Returns

bool

Example

```
ia.isconform("my_mystery.image")
```

regionmanager-Tool.html

1.1.2 regionmanager - Tool

Create and manipulate regions of interest
Requires:

Synopsis

Description

Overview of Regionmanager functionality

- Create simple pixel-coordinate based regions with functions

  box

  quarter

  Create simple world-coordinate based regions with functions

  frombcs (range along one axis)

  wrange (range along one axis)

  wbox

  wpolygon

  wdbox

  wdpolygon

  Also related is function setcoordinates

- Convert pixel regions to world regions with

  pixeltoworldregion.

- There are some general utility functions, generally only of interest if you are writing scripts. These are

  absreltype

  def

  done

  extractsimpleregions

  ispixelregion

  isworldregion

type

- There are some functions relating to interactive creation of regions with the Viewer. These are generally only of interest if you are writing scripts, and are

  displayedplane

  psuedotoworldregion

- There are some functions relating to communications. These are generally only of interest if you are writing scripts and are

  setselectcallback

  getselectcallback

Regions and the Regionmanager

When working with an image, one is generally interested in some part of that image on which astrophysical analysis is performed. This `region-of-interest` (or more generically and simply, the 'region') might be the whole image, or some subset of it.

Regions come in a few varieties. There are simple regular shapes (box, ellipsoid), simple irregular shapes (polygon), as well as compound regions made by combining other regions (simple or compound). For example unions or intersections of regions. In addition, the simple regions can be pixel-coordinate based or world-coordinate based. However, a compound regions must always comprise world-coordinate based regions.

It is the task of the Regionmanager (`tool`) to manage your regions; it creates, lists and manipulates them. Apart from a Regionmanager, the only other way to create a `region-of-interest` is with the viewier `tool`(type viewer within the casapy environment). This allows you to interactively make, with the cursor and an image display, a box or polygonal region. The region so made may be collected by the Regionmanager (in fact the complete process can be initiated from the Regionmanager).

The Regionmanager has a command line interface, but there are plans to have it interact directly with the CASA viewer`tool`. in the future. Currently the only way to interact is to save the regions created with the viewer to a file or as a region in the image , and use the Regionmanager function fromfiletorecord or fromtabletorecord, repsectively, to bring the regions in the CLI.

It is probably fair to say that for the simplest regions, such as a pixel box, there is little to choose between making a region with the GUI or the command line interface. However, for the world regions, the GUI is significantly better; it hides the details of handling the coordinate information.

Simple Pixel and World regions

Pixel regions are specified purely in terms of pixel coordinates. Pixel coordinates are considered to run from 1 at the bottom left corner (blc) of an image to the image shape at the top-right corner (trc).

For example, a pixel region might be box that runs from a bottom blc of [20,20] to a trc of [64,90] in a 2D image. Such a pixel region is NOT very portable though. If you are interested in a region about an astronomical source, pixel coordinates are only useful for a particular image. In some other image, the source may well have different pixel coordinates. However, pixel regions are easy to make and use, and they are valuable.

So far, we have been talking about absolute pixel coordinates. However, pixel regions can also be specified in relative coordinates. This is controlled through the argument `absrel` which may take on one of the values 'abs' (absolute coordinates) 'relref' (relative to the reference pixel of the image) or 'relcen' (relative to the center of the image). You can use the summary function of the image module to see the reference pixel. The sense of the offset is $rel = abs - ref$.

You may also define pixel regions in terms of fractional coordinates that are in the range [0,1] (blc to trc) - look for argument `frac` which can be T or F.

From the command line, let's make a simple pixel box region with

- ia.open('myimage') - csys = ia.coordsys() - r1 = rg.box(blc=[10,20], trc=[30,40]) - ia.statistics(region=r1); Number points = 441 Flux density = -1.728895e-03 Jy Sum = -9.522969e-02 Mean = -2.159403e-04 Variance = 1.518106e-05 Sigma = 3.896288e-03 Rms = 3.897854e-03

Minimum value is -1.208747e-02 at [13, 32] Maximum value is 1.287862e-02 at [21, 37]

You have now created a region tool called **r1** which describes a 2D box from [10,20] to [30,40] in absolute pixel coordinates. The region is made 'const' as well so you can't overwrite it by mistake. We then apply it to an image and evaluate some statistics in the specified region.

World regions are more portable, because the region is specified in terms of the world coordinates (e.g. RA and DEC, or Frequency etc). You can create a region with one image, and then apply it to another. For example, you might make an RA/DEC world box (i.e. blc and trc specified as RA and DEC) from an optical image. You can then apply it to your radio image of the same source. The software will look for the axes of the region (in this case RA and DEC), and then attempt to match them with the image to which the region is being applied. For each matching axis, the region will be applied.

When you make a world region, you must supply a coordinate system from the appropriate image to the Regionmanager. The coordinate system comes from the coordsys function of the Image tool. This associates a coordinate with an image pixel axis.

Now let's do an example with a world box. It is better to make a world box with the GUI interface, but the command line is still manageable. First, let's summarise the header of our image (the one from the previous example).

- ia.open('myimage') - ia.summary() Image name : myimage Image mask : Absent Image units : JY/BEAM

Direction system : J2000

Name Proj Shape Tile Coord value at pixel Coord incr Units

——————————————————————— Right Ascension SIN

155 155 17:42:29.303 90.00 -1.000000e+00 arcsec Declination SIN 178 178 -28.59.18.600 90.00 1.000000e+00 arcsec T
And now we will make a world box with the blc at the reference pixel, and the trc offset somewhat. We use the quanta module for creating associations of values and units; in particular, the quantity function.
- blc = "17:42:29.303 -28.59.18.600"; - trc = "17:42:28.303 -28.59.10.600"; - csys = ia.coordsys() - r2 = rg.wbox(blc=blc, trc=trc, pixelaxes=[0,1], csys=csys.torecord()) - ia.boundingbox(r2); [blc=[90 90], trc=[103 98], inc=[1 1], bbShape=[14 9], regionShape=[14 9], imageShape=[155 178]]
As well as passing in the blc and trc quantities, we have also told the Regionmanager which axes these values pertain to (they are numbered in the order in which they list in the `summary` function), and we also supplied it with the coordinate system of the image. Note that with the GUI interface, both the axes and coordinates handling is hidden. If we wished, we could have left out the `pixelaxes` argument, and it would have defaulted to [0,1]. We then find the bounding box of the region when it's applied to the image. You can see from the summary listing that it is correct.
We generally think about our images as being in some order; RA/DEC/Frequency (for the X/Y/Z axes) or whatever. In the application of world regions, this order is unimportant. If you created a world region from an RA/DEC/Frequency image, and applied it to a Frequency/DEC/RA image, that image order change would be accounted for.
Note that for pixel regions however, because the region is not tagged with a coordinate system, the order of the image is relevant. So if you made a pixel box from an RA/DEC image and applied it to a DEC/RA image, the RA range would be applied to the DEC axis and vice versa. The pixel regions are present for simple usage only.

Simple pixel regions as world regions
It is possible to create a world region that is specified in pixel coordinates, and maintains the coordinate axis information. This can be useful because compound regions must be world-coordinate based (see below). We can do this because we have defined in the Regionmanager, "new" world units, called "pix" (which are then known to the quanta module) and these are recognized in the world regions. Similarly, we have also defined a unit called "frac". This allows you to specify world coordinates in units of the fraction of the image shape. For example, the blc of a 2D image has a "frac" coordinate of [0,0] and a trc of [1,1]. But be careful, such regions are still not portable to another image, because they are still effectively pixel-coordinate based (although masquerading as world regions).
- ia.maketestimage() - csys = ia.coordsys() - - r1 = rg.wbox(blc="10pix,10pix", trc="20pix,20pix", pixelaxes=[0,1], csys=csys.torecord()) - localstats = ia.statistics(region=r1, list=False, verbose=False) - print 'Region 1: number points = ', localstats['npts'][0]
Region 1: number points = 121 - - r2 = rg.wbox(blc="30pix 30pix", trc="40pix 40pix", pixelaxes=[0,1], csys=csys.torecord()) -

localstats=ia.statistics(region=r2, list=False, verbose=False) - print 'Region 2: number points = ', localstats['npts'][0] Region 2: number points = 121
- regions= - regions['r1']=[r1] - regions['r2']=[r2] - r3 = rg.makeunion(regions) - localstats=ia.statistics(region=r3, list=False, verbose=False) - print 'Region 3: number points = ', localstats['npts'][0] Region 3: number points = 242
In this example we make a union from two pixel boxes masquerading as world regions.

Compound Regions

It is often desirable to make a region which combines others. These are called compound regions. For example, give me the union of 3 regions and evaluate the statistics in that union. Or intersect this region with that one and extract the image data from that region. Compound regions are fully recursive; you can nest them as deeply as you like.

You *must* be aware that compound regions can only be made from world-coordinated based regions. You can convert a pixel region to a world region with the pixeltoworldregion function (at some point this function will move from the Regionmanager `tool` to the Image `tool`.

- ia.open('myimage') - csys ia.coordsys(); - - blc = "17:42:29.303, -28.59.18.600"; - trc = "17:42:28.303, -28.59.10.600"; - r1 = rg.wbox(blc=blc,trc=trc,pixelaxes=[0,1],csys=csys.torecord()); - ia.boundingbox(r1); [blc=[90 90] , trc=[103 98] , inc=[1 1], bbShape=[14 9] , regionShape=[14 9] , imageShape=[155 178]] - - blc = "17:42:27.303, -28.59.18.600" - trc = "17:42:26.303, -28.59.10.600" - r2 = rg.wbox(blc=blc,trc=trc,pixelaxes=[0,1],csys=csys.torecord()); - ia.boundingbox(r2); [blc=[116 90] , trc=[129 98] , inc=[1 1], bbShape=[14 9] , regionShape=[14 9] , imageShape=[155 178]] - - regions= - regions['r1']=[r1] - regions['r2']=[r2] - r3 = rg.makeunion(regions) - bb = ia.boundingbox(r3) - bb [blc=[90 90] , trc=[129 98] , regionShape=[40 9] , imageShape=[155 178]] - - s1 = ia.statistics(region=r1, list=False) - s2 = ia.statistics(region=r2, list=F) - s3 = ia.statistics(region=r3, list=F) - - np = (bb['trc'][0]-bb['blc'][0]+1)*(bb['trc'][1]-bb['blc'][1]+1) - print 'No. pts in r1, r2, r3, bb(r3) = ', s1['npts'][0], s2['npts'][0], s3['npts'][0], np No. pts in r1, r2, r3, bb(r3) = 126, 126, 252, 360 - - - r4 = rg.intersection(r1, r2) - ia.boundingbox(r4) - r4

We make two world boxes and find the bounding box of each. Then we make the union of the two and find the bounding box which we store in a record called `bb`. We then find some statistics over the union (supressing the logger output and storing the results in records). You can see that the number of points found in the region reflects the union, not the bounding box of the union.

Finally, we make an intersection of our two regions. Because regions r1 and r2 don't actually intersect, so the region returned is empty.

Automatic Extension

One general philosophy behind the regions software is that if you apply a region to an image, but don't explicitly specify all axes of the image in that

region, then for the unspecified axes, the full image is taken. For example, imagine that from an RA/DEC optical image you made an RA/DEC region. You then applied it to an RA/DEC/Frequency radio spectral-line cube. For the frequency axis, which was not specified in the region, all frequency pixels would be selected. Another philosophy is that if, on application, a region extends beyond an image, the overhang is silently discarded.
- ia.open('myimage') - ia.boundingbox(rg.box()) [blc=[1 1] , trc=[155 178] , inc=[1 1] , bbShape=[155 178] , regionShape=[155 178] , imageShape=[155 178]] - - r1 = rg.box(trc=[20]) - ia.boundingbox(r1) [blc=[1 1] , trc=[20 178] , inc=[1 1] , bbShape=[20 178] , regionShape=[20 178] , imageShape=[155 178]] - - r2 = rg.box(trc=[9000,20]) - ia.boundingbox(r2) [blc=[1 1] , trc=[155 20] , inc=[1 1] , bbShape=[155 20] , regionShape=[155 20] , imageShape=[155 178]]
In the first example, none of the axes are specified in the region, so it defaults to the full image on all axes. In the second example only the first axis of the trc is specified. In the third example, the trc overhang is silently discarded.

Masks
Some comment about the combination of masks and `regions-of-interest` is useful here. Consider a simple polygonal region. This `region-of-interest` is defined by a bounding box, the polygonal vertices, and a mask called a `region mask`. The `region mask` specifies whether a pixel within the bounding box is inside or outside the polygon. For a simple box `region-of-interest`, there is obviously no need for a `region mask`.
Now imagine that you wish to recover the mask of an image from a polygonal `region-of-interest`. Now, necessarily, the mask is returned to you in regular Boolean array. Thus, the array shape reflects the bounding-box of the polygonal region. If the actual `pixel mask` that you apply is all good, then the retrieved mask would be good inside of the polygonal region and bad outside of it. If the actual `pixel mask` had some bad values in it as well, the retrieved mask would be bad outside of the polygonal region. Inside the polygonal region it would be bad if the `pixel mask` was bad.
More simply put, the mask that you recover is just a logical "and" of the `pixel mask` and the `region mask`; if the `pixel mask` is T *and* the `region mask` is T then the retrieved mask is T (good), else it is F (bad).

Vector Inputs
Many of the functions of a Regionmanager `tool` take vectors (numeric or strings) as their arguments. You can assume that

- For numeric vectors you can enter as an actual numeric vector (e.g. ['1.2','2.5','3']), a vector of strings (e.g. "1.2 2.5 3") or even a string with white space and/or comma delimiters (e.g. '1.2 2.5 3').
- For string vectors you can enter as an actual vector of strings (e.g. "1.2 2.5 3") or a string with white space or comma delimiters (e.g. '1.2 2.5 3').

Default Values

When specifying blc and trc vectors for some regions (usually boxes) it can often be that you wish to default one axis but not others.
For example, you may like to specify a value for axis 2 but leave axis 1 at its defaults. This is trivial with the GUI interface (you just don't fill in the ones you are not interested in), but is a little harder with the command line interface.
There is a function called def which provides a magic value which can be used with pixel boxes for this purpose. For example,
THIS EXAMPLE IS NOT VALID YET
- ia.fromshape('x', [20,30]) - r = rg.box(trc=[rg.dflt(),10]) - ia.boundingbox(region=r) [blc=[1 1] , trc=[20 10] , regionShape=[20 10] , imageShape=[20 30]]
You can see that trc for axis 1 has defaulted to the shape of the image.
With world boxes, we have defined a magic unit called 'default' instead. Thus
- ia.fromshape('x', [20,30]) - csys = ia.coordsys() - rg.setcoordinates(csys.torecord()) - r = rg.wbox(trc="0dflt 10pix") - ia.boundingbox(region=r) [blc=[1 1] , trc=[20 10] , regionShape=[20 10] , imageShape=[20 30]]

Error Checking
Although some error checking is done when the region is created (in CASA), much of it does not occur until the region is applied to an image. In particular, when creating a compound region, it is *not* checked that the compound region contains only world-coordinate based regions. It is only when you apply the region to an image that these checks are made. Any errors that occur will cause exceptions to be generated. Thus, when writing scripts, you should always use try/except blocks as in this example.
- def getStats(imageobject, blc, trc, csys): try : r1 = rg.wbox(blc=blc, trc=trc, csys=csys.torecord()) imageobject.statistics(r1) except Exception, instance: print 'ERROR: ", instance return False
return True
This fairly silly function detects if the region creation failed (function given invalid arguments) and displays the exception if it did. If all is well statistics are then evaluated from the region, and True is returned.

How a region is stored in CASA
In CASA, a region is itself a `tool`. This means it can be transmitted about with *Glish*, saved to Tables and restored from Tables. A region is actually made with a generic container `tool` called an Itemcontainer. The regions that you make have no intelligence regarding their greater purpose in life (although of course an Itemcontainer does have some functions so they can be manipulated). All of the knowledge about regions lies with the Regionmanager. It should not be necessary for you to know anything about the insides of regions. You just need to know how to make them and how to use them.
THIS EXAMPLE IS NOT VALID ANYMORE

- r1 = rg.box([10,20], [30,40]) - r1 [gui=¡function¿, done=¡function¿, get=¡function¿, has_item=¡function¿, makeconst=¡function¿, length=¡function¿, names=¡function¿, set=¡function¿, add=¡function¿, fromrecord=¡function¿, torecord=¡function¿, type=¡function¿] - - r1.torecord() [name=LCSlicer, isRegion=3, blc=[10 20] , trc=[30 40] , inc=, fracblc=[F F] , fractrc=[F F] , fracinc=, arblc=[1 1] , artrc=[1 1] , oneRel=T, comment=]
First we make a simple pixel-box region. We then try and print it out. You can see that this isn't very enlightening. What you see are the Itemcontainer's functions. To really see inside the region, you can use the torecord function of Itemcontainer. This converts the region to a record which you can view if you really must (the record is of no functional use when dealing with regions).
It's perhaps also useful to know that once you have made a region you can't break it. You can delete it, maybe overwrite it, but you can't mistakenly mess up its contents (even with the Itemcontainer functions with which it was created). It's user proof - there's a challenge for you !

Methods

regionmanager	Construct a regionmanager
absreltype	Convert region type value to a string
box	Create a pixel box region
frombcs	Create a world coordinate region based on box-chan-stokes input
complement	Create the complement of a world region
concatenation	Concatenate world regions along a new axis
deletefromtable	Delete regions from a Table
difference	Create the difference of two world regions
done	Destroy this regionmanager
selectedchannels	Get an array of zero-based selected channel numbers from an input string specificaiton.
fromtextfile	Create a region dictionary from a region text file.
fromtext	Create a region dictionary from a region text string.
fromfiletorecord	Create a region record(s) from a file(s).
tofile	Create a region record file that can be read by from filetorecord.
fromrecordtotable	Save regions stored in a record into a Table
fromtabletorecord	Restore regions from a Table to a record
intersection	Create the intersection of some world regions
ispixelregion	Is this region a pixel region ?
isworldregion	Is this region a world region ?
namesintable	Find the names of the regions stored in a Table
setcoordinates	Set new default Coordinate System
makeunion	Create a union of world regions
wbox	Create a world box region
wpolygon	Create a world polygon region with quantities

regionmanager.regionmanager.html

regionmanager.regionmanager - Function

1.1.2 Construct a regionmanager

Description

This is the only Regionmanager constructor. It should generally be unnecessary for you to make one as there is little state in a Regionmanager (you can set a Coordinate System with setcoordinates); the default Regionmanager **rg** should be all you need.

Arguments

Returns
casaregionmanager

---

regionmanager.absreltype.html

regionmanager.absreltype - Function

1.1.2 Convert region type value to a string

Description

This function is not intended for general user use.
Regions may be specified with coordinates which are absolute or relative. This function converts the integer code defining the absolute/relative type of the coordinates (which is stored in the region) into a string (maybe for printing purposes).
The different types are
Integer String Description 1 abs Absolute coordinate 2 relref Relative reference pixel 3 relcen Relative to center of image 4 reldir Relative to some direction

Arguments

Inputs		
absrelvalue	Region type value	
	allowed:	int
	Default:	0
		Integer - 1, 2, 3, 4

Returns
string

Example

```
- r = rg.box(blc=[3,40], trc=[80,90])   # Create region
- v = r.get('arblc')                    # Get absrel value vector for blc
- for i in range( len(v) ):
+    print rg.absreltype(v[i])          # Print string conversion for each axis
-
```

regionmanager.box.html

regionmanager.box - Function

1.1.2 Create a pixel box region

Description

This function creates a multi-dimensional pixel box region. The box is specified by a bottom-left corner, and top-right corner and an increment (or stride). Pixel coordinates are considered to run from 1 at the bottom left corner of the image to the image shape at the top-right corner of the image. You can specify whether the coordinates are given as pixel coordinates (`frac=F`) or fractions of the image shape (`frac=T`). Absolute fractions are in the range [0,1].
You can also specify whether the coordinates are given as absolute coordinates (`absrel='abs'`) or relative to the reference pixel (`absrel='relref'`) or relative to the center of the image (`absrel='relcen'`).

Arguments

Inputs		
blc	blc of the box	
	allowed:	doubleArray
	Default:	0
		Unity
trc	trc of the box	
	allowed:	doubleArray
	Default:	-1
		Shape
inc	increment	
	allowed:	doubleArray
	Default:	1
absrel	Absolute or relative coordinates	
	allowed:	string
	Default:	relref
		relcen
		abs
frac	Pixel or fractional coordinates	
	allowed:	bool
	Default:	false
		F
comment	A comment stored with the region	
	allowed:	string
	Default:	

Returns
record

Example

```
ia.open('myimage')
ia.shape()
[155 178 256]

r = rg.box()                  # create region
-
- ia.boundingbox(r)
[blc=[1 1 1] , trc=[155 178 256] , inc=[1 1 1] , bbShape=[155 178 256] ,
regionShape=[155 178 256] , imageShape=[155 178 256] ]
```

This region, on application to an image, selects the entire image.

Example

```
- ia.open('myimage')
- ia.shape()
[155 178 256]
-
- r=rg.box(blc=[5,10])
- ia.boundingbox(r)
[blc=[5 10 1] , trc=[155 178 256] , inc=[1 1 1] , bbShape=[151 169 256] ,
regionShape=[151 169 256] , imageShape=[155 178 256] ]
```

This region is only specified for the first two axes of the blc.
Automatic extension rules apply for the other axis and the trc
(defaults to the shape).

Example

```
- ia.open('myimage')
- ia.shape()
[155 178 256]
-
- r = rg.box(blc=[10, 10, 10], trc=[20, 20, 20], inc=[2, 2, 2])
- ia.boundingbox(r)
[blc=[10 10 10] , trc=[20 20 20] , inc=[2 2 2] , bbShape=[11 11 11] ,
regionShape=[6 6 6] , imageShape=[155 178 256] ]
-
stats=ia.statistics(region=r, list=False);
stats['npts'][0]
216
```

This region picks out every other pixel in the 3D box. The ‘‘regionShape’’ field of the bounding box record does reflect the increment whereas ‘‘bbShape’’ does not. You can see that the number of points used in determining the statistics (216) reflects the increment as well.

Example

THIS EXAMPLE IS NOT VALID YET

```
- ia.open('myimage')
- ia.shape()
[64 128]
-
- rmd = rg.dflt()
- r = rg.box([-5,-10], [rmd,20], absrel='relcen')
-
- ia.boundingbox(r)
[blc=[28 55] , trc=[64 85] , inc=[1 1] , bbShape=[37 31] ,
regionShape=[37 31] , imageShape=[64 128] ]
```

The region is specified in pixels relative to the center of the image. Note the use of the default value ({\cf rg.dflt()}) to default the first axis of the trc argument to the image shape without having to know the image shape.

Example

```
- ia.open('myimage')
- ia.shape()
[155 178 256]
-
```

```
- summ=ia.summary(list=False)
- summ['header']['refpix']
[90 90 1]
-
- r = rg.box([-0.25,-0.3], [0.25, 0.5], frac=True, absrel='relref')
- ia.boundingbox(r)
[blc=[39 37 1] , trc=[116 178 256] , inc=[1 1 1] , bbShape=[78 142 256] ,
regionShape=[78 142 256] , imageShape=[155 178 256] ]
```

This example shows selection by relative to reference pixel fractional coordinates plus auto extension to unspecified axes.

---

regionmanager.frombcs.html

regionmanager.frombcs - Function

1.1.2 Create a world coordinate region based on box-chan-stokes input

Description

This function creates a multi-dimensional world coordinate region based on box, chans, stokes inputs familiar from image analysis tasks. It is being introduced as a temporary means of refactoring some python level task code into C++. However, if users find it to have value, its existence can be permanent.

Arguments

Inputs	
csys	Coordinate system record. Must be specified. allowed: record Default:
shape	shape of the image. Necessary for boundedness checks. Must have the same number of dimensions as the associated coordinate system. Default = [] allowed: intArray Default: 0
box	Direction plane box specification as normally provided in image analysis tasks. "" means use entire directional plane as specified in shape. Default "". allowed: string Default:
chans	Channel spec as normally provided to image analysis tasks. "" means use all channels, Default "". allowed: string Default:
stokes	Stokes spec as normally provided to image analysis tasks. "" means use stokescontrol for setting stokes. Default "". allowed: string Default:
stokescontrol	Polarization set to use if stokes parameter is not specified. Choices are "a" (use all stokes) and "f" (use first stokes). Default "a". allowed: string Default: a
region	Named region in the form imagename:regionname or region dictionary. Used only if box, chans, stokes not specified. Default "". allowed: any Default: variant

Returns
record

Example

regionmanager.complement.html

regionmanager.complement - Function

1.1.2 Create the complement of a world region

Description

This function (short-hand name `comp`) creates the complement of a world region(s).
The region parameter can be a single region record defining a simple or complex region or it can contain several region records in a Python dictionary. If multiple regions are given then the union of this set of regions is taken first, and the complement is found from the union.
NOTE: ia.statistics() is UNABLE to handle complement regions in CASA yet.

Arguments

Inputs		
region	The world region	
	allowed:	any
	Default:	variant
		Region tool
comment	A comment stored with the region	
	allowed:	string
	Default:	String

Returns
record

Example

```
- ia.open('hcn')
- csys = ia.coordsys()
- ia.shape()
[155 178]
-
```

```
- blc = "17:42:29.303 -28.59.18.600"
- trc = "17:42:28.303 -28.59.10.600"
- r2 = rg.wbox(blc,trc,[1,2],csys.torecord())
- r3 = rg.complement(r2);
-
- ia.statistics(region=r2)                          # Some output discarded
Selected bounding box [90, 90] to [103, 98]
No pts   = 126
-
- ia.statistics(region=r3)
Selected bounding box [1, 1] to [155, 178]     # Some output discarded
No pts   = 27464
```

As expected, the number of pixels in the complement is $(155*178)-126=27464$

---

regionmanager.concatenation.html

regionmanager.concatenation - Function

1.1.2 Concatenate world regions along a new axis

Description

This function (short-hand name `concat`) creates a region which is the concatenation along a new axis of the given world regions.
This function is similar to the extension function. The `concatenation` function allows you to take many world regions, and concatenate them along one axis (rather than take one region and extend it along many axes which is what function `extension` does).
For example, you may have generated a different polygonal region for each spectral pixel of a spectral-line cube and you wish to concatenate them together to form the overall region for use in a deconvolution application.
The axis to concatenate along is specified as a 1-dimensional world box. The shape of the 1D box must contain as many pixels (although you don't have to specify it in pixels) as there are regions to concatenate.
Because this function is most likely to be used in a script, the interface takes a record containing `region` records, Python dictionaries, as there might be a lot of them.

Arguments

Inputs		
box	The axis to concatenate along	
	allowed:	any
	Default:	variant
		world box region
regions	World regions	
	allowed:	any
	Default:	variant
		Record containing world regions
comment	A comment stored with the region	
	allowed:	string
	Default:	String

Returns
record

Example

```
- ia.open('cube')
- csys = ia.coordsys()
- rg.setcoordinates(csys.torecord(), verbose=False) # Don't tell us each time
                                                    # private coordinates used
- box = rg.wbox(blc="20pix", trc="25pix", pixelaxes=[2])
- bb = ia.boundingbox(box)
-
- regs = {};
- local x, y;
- for i in bb.blc[3]:bb.trc[3]:
+ # Some code in function 'mypolygon' generates the
+ # x and y vectors for this spectral pixel, perhaps interactively
+
+    mypolygon(x,y);
+    regs["reg"+str(j)] = rg.wpolygon(x,y,[0,1])
- rc = rg.concatenation(box, regs)
-
- ia.statistics(region=rc, axes=[1,2])
Plane Freq         Npts Sum            Mean           Rms            Sigma          Minimum
20    1.413724e+09 25   -4.778154e+00  -1.911262e-01  2.578399e-01   1.766359e-01   -4.25243
21    1.413744e+09 40   -7.476902e+00  -2.990761e-01  3.692736e-01   2.210687e-01   -6.07364
22    1.413763e+09 32   -2.696485e+00  -1.078594e-01  1.916686e-01   1.617070e-01   -3.29578
23    1.413783e+09 77   4.889158e-01   1.955663e-02   3.148451e-02   2.518293e-02   -3.67194
24    1.413803e+09 25   -1.337832e+00  -5.351327e-02  6.296221e-02   3.385893e-02   -1.23249
25    1.413823e+09 15   1.091297e+00   4.365189e-02   7.252339e-02   5.910932e-02   -6.36456
```

In this example, we create a 1D box and use it to concatenate 2D xy polygons along the z axis. We then ask for the statistics of each plane in the region. There is a differerent number of pixels per plane as each polygon is different.

---

regionmanager.deletefromtable.html

regionmanager.deletefromtable - Function

1.1.2 Delete regions from a Table

Description

This function deletes a region stored in an casa Table.
For the `tablename` argument,
you have to give the name of an existing CASA table on disk (any kind of table).
You specify the name of the region with the `regionname` arguments. If you set `regionname=''` then nothing is done. The names of all the regions stored in a Table can be found with the function namesintable.

Arguments

Inputs		
tablename	The table	
	allowed:	string
	Default:	Image tool, table tool or String
regionname	Name(s) of the region(s) to delete	
	allowed:	string
	Default:	Vector of strings

Returns
bool

Example

```
- names = rg.namesintable(hcn)
- rg.deletefromtable(img, names[0])
```

In this example, we delete the first region that is reported to be in the Table {\tt 'hcn'}.

regionmanager.difference.html

regionmanager.difference - Function

1.1.2 Create the difference of two world regions

Description

This function (short-hand name `diff`) creates a region which is the difference of two world regions. The order of the regions is important.
The difference consists of all pixels masked-on in the first region and not masked-on in the second region.

Arguments

Inputs		
region1	The first world region	
	allowed:	record
	Default:	
region2	The second world region	
	allowed:	record
	Default:	Region tool
comment	A comment stored with the region	
	allowed:	string
	Default:	String

Returns
record

Example

```
- ia.open('hcn')
- csys = ia.coordsys()
- rg.setcoordinates(csys.torecord())
-
- blc = "10pix 10pix"
- trc = "60pix 60pix"
```

```
- r1 = rg.wbox(blc,trc,[0,1])
-
- blc = "50pix 50pix"
- trc = "80pix 80pix"
- r2 = rg.wbox(blc, trc, [0,1])
-
- r3 = rg.difference(r1, r2)                          # r1 - r2
-
- ia.statistics(region=r1)                            # Some output discarded
Selected bounding box [10, 10] to [60, 60]
No pts   = 2601
-
- ia.statistics(region=r3)                            # Some output discarded
Selected bounding box [10, 10] to [60, 60]
No pts   = 2480
-
-
- r4 = rg.difference(r2, r1)                          # r2 - r1
-
- ia.statistics(region=r2)                            # Some output discarded
Selected bounding box [50, 50] to [80, 80]
No pts   = 961
-
- ia.statistics(region=r4)                            # Some output discarded
Selected bounding box [50, 50, 1] to [80, 80, 64]
No pts   = 840
```

We use pixel units and boxes in this example to make it clear what is happening. The two regions overlap in the top right corner area of region {\stf r1} by an area of $11\times11=121$~pixels. Therefore, the difference region {\stf r3} has $2601-121=2480$~pixels in it. For difference region {\stf r4}, the region of overlap is the bottom left corner area of region {\stf r2} and still contains 121~pixels. We expect $961-121=840$~pixels in the differnce region.

---

regionmanager.done.html

regionmanager.done - Function

1.1.2 Destroy this regionmanager

Description

This function destroys the contents of the **regionmanager** `tool` (including its GUI). The `tool` still exists as a Glish variable, but it is no longer a Regionmanager ! You are unlikely to need this function.

Arguments

Returns
bool

---

regionmanager.selectedchannels.html

regionmanager.selectedchannels - Function

1.1.2 Get an array of zero-based selected channel numbers from an input string specificaiton.

Description

This method returns all the selected zero-based channel numbers from the specified string within the image.

Arguments

Inputs	
specification	Valid channel specification. See help par.chans for examples. allowed: string Default:
shape	Image shape. Used to determine if the specificaiton lies outside the image. allowed: intArray Default: 0

Returns
intArray

Example

```
ia.fromshape("",[20,20,20])
rg.setcoordinates(ia.coordsys().torecord())
selected_channels = rg.selectedchannels(specification="range=[40km/s,50km/s]", shape
ia.done()
```

regionmanager.fromtextfile.html

regionmanager.fromtextfile - Function

1.1.2 Create a region dictionary from a region text file.

Description

This function reads a text file containing region descriptions and converts it to a python dictionary.

Arguments

Inputs		
filename	List of text file containing the region description	
	allowed:	string
	Default:	
shape	Image shape.	
	allowed:	intArray
	Default:	0
csys	Coordinate system record. Defaults to coordinate system used in rg.setcoordinates()	
	allowed:	record
	Default:	

Returns

record

---

regionmanager.fromtext.html

regionmanager.fromtext - Function

1.1.2 Create a region dictionary from a region text string.

Description

This function reads a region region text descriptions and converts it to a python region dictionary.

Arguments

Inputs		
text	region description	
	allowed:	string
	Default:	
shape	Image shape, only used if first region is a difference.	
	allowed:	intArray
	Default:	1
csys	Coordinate system record. Defaults to coordinate system used in rg.setcoordinates()	
	allowed:	record
	Default:	

Returns

record

Example

```
ia.open('test.image')
csys=ia.coordsys()
rg.setcoordinates(csys.torecord())
a=rg.fromtext("ellipse [[04h31m38.44139, 18d13m57.0861], [1.0arcsec, 1.0arcsec], 0.00000000
ia.done()

In this example, we create a circular region of 1 arcsec radius centered on J2000 04h31m38.4
```

regionmanager.fromfiletorecord.html

regionmanager.fromfiletorecord - Function

1.1.2 Create a region record(s) from a file(s).

Description

This function reads files containing ImageRegion objects and turns them into Region Records.
The intended use for this method is to read the file saved by the casa viewer and turn the files contents into regions that are usabla by the image analysis tool.

Arguments

Inputs		
filename	List of files containing the Image Regions	
	allowed:	string
	Default:	File name(s)
verbose	Report successful saves	
	allowed:	bool
	Default:	true
		true
regionname	Name(s) of the region(s) when saved in the table	
	allowed:	string
	Default:	Self naming

Returns
record

Example

```
- img = ia.open('hcn')
- rg.fromfiletorecord(T, "x1 x2", "file1, file2", r1, r2)
- ia.statistics( region=r1, verbose=True )
- ia.statistics( region=r2, verbose=True )
```

In this example, we create two regions called {\stf r1} and {\stf r2}
from the files ???
The regions are renamed to ‘x1’ and ‘x2’ as they are stored.

Example

```
e
- img = ia.open('hcn')
- r1 = rg.box()f
- r2 = rg.quarter()
- rg.fromglobaltotable(img, T, F, "", r1, r2)
- rg.namesintable(img)
x1 x2
```

In this example, we save two regions called {\stf r1} and {\stf r2} to
the table (previously containing no regions) referred to by the image
tool {\stf im}. The names for regions are made up for us as we don't
specify them. Note that because the regions are specified by the
special \glish\ ‘...’ argument (it has no actual argument name), we
must give the {\stfaf regionname} argument explcitly as an empty vector
of strings (else \glish\ will take the empty string as a region).

---

regionmanager.tofile.html

regionmanager.tofile - Function

1.1.2 Create a region record file that can be read by from filetorecord.

Description

This function is to store a region created by the regionmanager in a disk file for future use

Arguments

Inputs		
filename	List of files containing the Image Regions	
	allowed:	string
	Default:	File name(s)
region	region record/dict to store in the file	
	allowed:	record
	Default:	

Returns
bool

Example

```
- img = ia.open('hcn')
- imcs=ia.coordsys()
- blc = ['16:28:25.50',  '+040.49.05.61']
- trc = ['16:24:28.67', '+041.45.28.43']
- r1 = rg.wbox(blc=blc,trc=trc,pixelaxes=[0,1],csys=imcs.torecord())
- rg.tofile('myboxregion', r1)
- r1readback=rg.fromfiletorecord('myboxregion')
```

```
In this example we create a box region using world coordinates for blc and trc. We save that
Then we read it back using the function rg.fromfiletorecord and store it in a variable {\tt
```

identical.

---

regionmanager.fromrecordtotable.html

regionmanager.fromrecordtotable - Function

1.1.2 Save regions stored in a record into a Table

Description

This function saves regions into an casa Table For the `tablename` argument the user should be the name of an existing CASA Table on disk (any kind of table). If the parameter `asmask` is `True` then the table has to be an image table. A mask makes sense with an image only.
You can specify the name the region will have (`regionname`) when it is saved in the Table. If you don't specify this, a digit based name is assigned to it or if specify a name that already exists a new one will be generated which is close but different. The function returns you the name the region is assigned

Arguments

Inputs		
tablename	The table	
	allowed:	string
	Default:	Image tool, table tool or String
regionname	Name(s) of the region(s) when saved in the table	
	allowed:	any
	Default:	variant
		Vector of strings
regionrec	Region(s) to save	
	allowed:	record
	Default:	Record of region tool(s)
asmask	save region as a mask rather than region	
	allowed:	bool
	Default:	false
		false
verbose	Report successful saves	
	allowed:	bool
	Default:	true
		True

Returns
string

Example

```
- ia.open('hcn')
- cs=ia.coordsys()
- blc = "16:28:25.50 +040.49.05.61"
- trc = "16:24:28.67 +041.45.28.43"
- r1 = rg.wbox(blc="10pix 20pix",trc="30pix 40pix",pixelaxes=[0,1],csys=cs.torecord())
- r2 = rg.wbox(blc=blc,trc=trc,pixelaxes=[0,1],csys=cs.torecord())
- rg.fromrecordtotable('hcn', "x", r1)
x
- rg.fromrecordtotable('hcn', "x", r2)
x0
- rg.namesintable('hcn')
x x0
```

Example

```
2 CASA image files on disk 'hcn1' 'hcn2'

- names = rg.namesintable('hcn1')
- r = rg.fromtabletorecord('hcn1', names[0])
- rg.namesintable('hcn2')
- rg.fromrecordtotable('hcn2', names[0], r)


In this example, we recover a region into a record
from one image, and then copy them to another.
```

Example

```
####In this example a region is saved as a mask
ia.open('myfancy.image')
csys=ia.coordsys()
ia.done()
##Lets make a world-box  region
wbox=rg.wbox(['10pix', '10pix', '0pix', '0pix'], ['20pix', '20pix', '0pix', '0pix'], csys=cs
###save that into the image as a mask rather than just a region and assign it the name
###mask1
rg.fromrecordtotable('myfancy.image', 'mask1', wbox, asmask=True)
###now let us set that as default mask
ia.open('myfancy.image')
ia.maskhandler('set', 'mask1')
ia.done()
###and now let us view that image
viewer('myfancy.image')
```

---

regionmanager.fromtabletorecord.html

regionmanager.fromtabletorecord - Function

1.1.2 Restore regions from a Table to a record

Description

This function restores a region from an CASA Table to the global name space. For the `tablename` argument, you can specify an image `tool`, a table `tool`, or a string. If you give a string, it should be the name of an existing CASA table on disk (any kind of table).
If `numberfields` is F, then the field names of the record are the same as they are in the Table. Otherwise, the regions are put into numbered fields (the field names could be anything).
You can use the function namesintable to find out the names of the regions in the Table.

Arguments

Inputs		
tablename	The table	
	allowed:	string
	Default:	Image tool, table tool or String
regionname	Name of the region(s) to restore	
	allowed:	any
	Default:	variant
		All
verbose	Report successful restores	
	allowed:	bool
	Default:	true
		True

Returns
record

Example

```
- img = ia.open('hcn')
- rec = rg.fromtabletorecord(img, numberfields=T)
- print is_region(rec[0])
```

The record fields are numbered, not named.

---

regionmanager.intersection.html

regionmanager.intersection - Function

1.1.2 Create the intersection of some world regions

Description

This function (short-hand name `int`) creates a region which is the intersection of the given world regions. The input regions can themselves be compound regions (such as the union or intersection etc). The input regions must be provided as a Python dictionary of regions (see examples).

Arguments

Inputs		
regions	World regions and comment	
	allowed:	any
	Default:	variant Region tools or record of region tools, and String
comment	A comment stored with the region	
	allowed:	string
	Default:	String

Returns
record

Example

```
- ia.open('hcn')
- csys = ia.coordsys()
- rg.setcoordinates(csys.torecord())
-
- blc = "10pix 10pix 1pix"
- trc = "60pix 60pix 1pix"
- r1 = rg.wbox(blc=blc, trc=trc, pixelaxes=[0,1,2])
```

```
-
- x = qa.quantity([50,55,58,65,58,53,50], 'pix')
- y = qa.quantity([50,53,69,70,63,58,55], 'pix')
- r2 = rg.wpolygon(x=x, y=y, pixelaxes=[0,1])
-
- regions= {'region1':r1, 'region2':r2}
- r3 = rg.intersection(regions, 'This is the comment')
-
- ia.boundingbox(r1)
[blc=[10 10 1] , trc=[60 60 256] , regionShape=[51 51 256] , imageShape=[155 178 256] ]
- ia.boundingbox(r2)
[blc=[50 50 1] , trc=[65 70 256] , regionShape=[16 21 256] , imageShape=[155 178 256] ]
- ia.boundingbox(r3)
[blc=[50 50 1] , trc=[60 60 256] , regionShape=[11 11 256] , imageShape=[155 178 256] ]
-
- ia.statistics(region=r3)                          # Some output discarded
NORMAL: Selected bounding box [50, 50, 1] to [60, 60, 1]
Number points = 51
```

In this example, we use pixel coordinates so that it is clear what is happening. You can see that the number of pixels in the intersection (51) is less than the number in the bounding box of the intersection (121) because the intersection is actually polygonal and does not fill the bounding box.

Example

```
- ia.open('onno')
- csys = ia.coordsys()
- x = qa.quantity([3,6,9,6,5,5,3],'pix')
- y = qa.quantity([3,4,7,9,7,5,5],'pix')
-
- regions = {};
- regions['poly'] = rg.wpoly(x,y,[1,2],csys.torecord())
-
- blc = "17:42:29.303 -28.59.18.600"
- trc = "17:42:28.303 -28.59.10.600"
- regions['box'] = rg.wbox(blc,trc,[0,1],csys.torecord())
```

```
-
- r3 = rg.intersection(regions,'The mysteries of CASA')
```

This example is the same as the prevoius one, except the regions are provided to the intersection function in a record, rather than directly in the call sequence.

---

regionmanager.ispixelregion.html

regionmanager.ispixelregion - Function

1.1.2 Is this region a pixel region ?

Description

NOT IMPLEMENTED IN CASA
This function returns T if the region is a pixel region. For any other *Glish* variable it returns F.

Arguments

Inputs		
region	The region	
	allowed:	record
	Default:	Region tool

Returns

bool

Example

```
- ia.open('hcn')
- csys = ia.coordsys()
- r1 = rg.box()                          # A pixel region
- r2 = rg.wbox(csys=csys.torecord())     # A world region
- rg.ispixelregion(r1)
T
- rg.ispixelregion(r2)
F
- x = [20,30]
- rg.ispixelregion(x)
F
```

regionmanager.isworldregion.html

regionmanager.isworldregion - Function

1.1.2 Is this region a world region ?

Description

NOT IMPLEMENTED IN CASA
This function returns T if the region is a world region. For any other *Glish* variable it returns F.

Arguments

Inputs		
region	The region	
	allowed:	record
	Default:	Region tool

Returns
bool

Example

```
- ia.open('hcn')
- csys = ia.coordsys()
- r1 = rg.box()                          # A pixel region
- r2 = rg.wbox(csys=csys.torecord())     # A world region
- rg.isworldregion(r1)
F
- rg.isworldregion(r2)
T
- x = [20,30]
- rg.isworldregion(x)
F
```

regionmanager.namesintable.html

regionmanager.namesintable - Function

1.1.2 Find the names of the regions stored in a Table

Description

This function returns the names of regions stored in an CASA Table.
For the `tablename` argument, you can specify a string; it should be the name of an existing CASA table on disk (any kind of table).

Arguments

Inputs		
tablename	The table	
	allowed:	string
	Default:	Image tool, table tool or String

Returns

stringArray

Example

```
- names=rg.namesintable('hcn')
- names
r1 poly2 int0
```

---

regionmanager.setcoordinates.html

regionmanager.setcoordinates - Function

1.1.2 Set new default Coordinate System

Description

This function allows you to (re)set the default Coordinate System used by the functions that make world regions. If you don't specifiy a Coordinate System when you make the world region, the default Coordinate System, if there is one, is used. The Coordinate System is stored in a **coordinates** `tool` and is created with the coordsys `tool function`.
Normally, the world region creating functions like wbox and wpolygon will issue a message each time the private Coordinate System is used. However, if you set `verbose=F` then this will not occur.

Arguments

Inputs		
csys	Default Coordinate System for use in world regions	
	allowed:	record
	Default:	Coordinate tool

Returns
bool

Example

```
- ia.open('quiqui')
- csys = ia.coordsys()
- rg.setcoordinates(csys.torecord())
- r1 = rg.wbox()
Using private CoordinateSystem from image "quiqui"
```

regionmanager.makeunion.html

regionmanager.makeunion - Function

1.1.2 Create a union of world regions

Description

This function takes a minimum of two world regions and creates a region which is the union of the given regions. The input regions can themselves be compound regions (such as the union or intersection etc). The input regions must be a Pythion dictionary of at leat two regions (see examples).

Arguments

Inputs		
regions	World regions and comment	
	allowed:	any
	Default:	variant
		Record/dict of regions to be unionized (the key names are immaterial)
comment	A comment stored with the region	
	allowed:	string
	Default:	

Returns
record

Example

```
- ia.open('onno')
- csys = ia.coordsys()
- x = qa.quantity([3,6,9,6,5,5,3],'pix')
- y = qa.quantity([3,4,7,9,7,5,5],'pix')
- r1 = rg.wpoly(x,y,[1,2],csys.torecord())
-
- blc = "17:42:29.303 -28.59.18.600"
```

```
- trc = "17:42:28.303 -28.59.10.600"
- r2 = rg.wbox(blc,trc,[0,1],csys.torecord())
-
- regions= {'region1':r1, 'region2':r2}
- r3 = rg.makeunion(regions,'The mysteries of CASA')
-
- ia.shape()
[155 178 256]
- ia.boundingbox(r1)
[blc=[3 3 1] , trc=[9 9 256] , inc=[1 1 1] , bbShape=[7 7 256] ,
regionShape=[7 7 256] , imageShape=[155 178 256] ]
- ia.boundingbox(r2)
[blc=[90 90 1] , trc=[103 98 256] ,  inc=[1 1 1] , bbShape=[14 9 256] ,
regionShape=[14 9 256] , imageShape=[155 178 256] ]
- ia.boundingbox(r3)
[blc=[3 3 1] , trc=[103 98 256] ,  inc=[1 1 1] , bbShape=[101 96 256] ,
regionShape=[101 96 256] , imageShape=[155 178 256] ]
-
- ia.statistics(region=r1)
Selected bounding box [3, 3, 1] to [9, 9, 256]
Number points = 6400
-
- ia.statistics(region=r2)
Selected bounding box [90, 90, 1] to [103, 98, 256]
Number points = 32256
-
- ia.statistics(region=r3)
Selected bounding box [3, 3, 1] to [103, 98, 256]
Number points = 38656
```

When the polygon only is applied, it is auto extended along the third axis. The {\stff statistics} function finds 6400 pixels in the region, which is $6400/256=25$ pixels per plane. Likewise, when the box only is applied, the {\stff statistics} function finds 32256 pixels in the region, which is $32256/256=126$ pixels per plane. When the union is applied, the {\stff statistics} function finds 38656 pixels in the region. First it finds the union of the polygon and box (which are specified only in the XY plane) and that union is extended. Thus we expect $(25+126)*256=38656$ pixels in the region of the union, as found.

Example

```
- ia.open('onno')
- csys = ia.coordsys()
- x = qa.quantity([3,6,9,6,5,5,3],'pix')
- y = qa.quantity([3,4,7,9,7,5,5],'pix')
-
- regions = {}
- regions['poly'] = rg.wpoly(x,y,[0,1],csys.torecord())
-
- blc = "17:42:29.303 -28.59.18.600"
- trc = "17:42:28.303 -28.59.10.600"
- regions['box'] = rg.wbox(blc,trc,[0,1],csys.torecord())
-
- r3 = rg.union(regions,'The mysteries of CASA')
```

This example is the same as the prevoius one, except the regions are provided to the union function in a record, rather than directly in the call sequence.

---

regionmanager.wbox.html

regionmanager.wbox - Function

1.1.2 Create a world box region

Description

This function creates a multi-dimensional world box region; the corners of the box are specified in world coordinates. However, the box is not a true world volume in that its sides do not follow world contours. Its sides are parallel to the pixel axes. If you are in a region of high world coordinate contour non-linearity (e.g. near a pole), you are probably better off using a world polygon.
The box is specified by a bottom-left corner, and a top-right corner. The coordinates are given as quantities, and you can give a vector of quantities (e.g. `blc = qa.quantity("1rad 20deg")` or a quantity of a vector (e.g.`blc = qa.quantity([10,30], 'rad')`).
You can specify whether the coordinates are given as absolute coordinates (`absrel='abs'`) or relative to the reference pixel (`absrel='relref'`) or relative to the center of the image (`absrel='relcen'`). You can specify this for each axis (the same for the blc and trc). If you specify less values than the number of values in `blc` or `trc` then the last value you did specify is used as the default for all higher numbered axes (e.g. `absrel='relref'` means `absrel="relref relref"` for two axes).
You specify which pixel axes in the image the `blc` and `trc` vector refer to with the `pixelaxes` argument. If you don't, it defaults to [0,1,2,...]. This specification is an important part of world regions.
You must also specify the Coordinate System with the `csys` argument. The Coordinate System is encapsulated in a `coordinates` tool and can be recovered from an image with the coordsys tool function. You can also set a default Coordinate System in the Regionmanager with the setcoordinates function.
In the Regionmanager we have defined units 'pix' and 'frac'; these are then known to the quanta system. This means that you can effectively define a pixel box (except for the stride capability) as a world box with most of the advantages of world regions (can be used for compound regions). However, it is still not very portable to other images because the coordinates are pixel based, not world based.
Note that the need to deal with the `pixelaxes` and `csys` is hidden from you when using the gui interface of the Regionmanager.

Arguments

Inputs		
blc	blc of the box ; a vector of quantities	
	allowed:	any
	Default:	variant
		Unity
trc	trc of the box; a vector of quantities	
	allowed:	any
	Default:	variant
		Shape
pixelaxes	Which pixel axes	
	allowed:	intArray
	Default:	-1
		[0,1,2,...]
csys	Coordinate System	
	allowed:	record
	Default:	Private Coordinate System
absrel	Absolute or relative coordinates Vector of strings from 'abs', 'relref' and 'relcen'	
	allowed:	string
	Default:	'abs'
comment	A comment stored with the region	
	allowed:	string
	Default:	

Returns
record

Example

```
- r = rg.wbox()
```

```
This region, on application to an image, will select the entire
image.
```

Example

```
- ia.open('ada')
- csys = ia.coordsys()
- csys.summary()

Name             Proj Shape Tile   Coord value at pixel    Coord incr Units
------------------------------------------------------------------------
Frequency                64   16  1.413350e+09     1.00  1.968717e+04 Hz
Velocity                          1.378053e+02     1.00 -4.174021e+00 km/s
Declination       SIN   178   89 -28.59.18.600    90.00  1.000000e+00 arcsec
Right Ascension   SIN   155   31  17:42:29.303    90.00 -1.000000e+00 arcsec
-
-
- blc = "17:42:29.303 -28.59.18.600"
- trc = "17:42:28.303 -28.59.10.600"
- r1 = rg.wbox(blc=blc,trc=trc,pixelaxes=[0,1],csys=csys.torecord())
- ia.boundingbox(r1)
[blc=[1 90 90] , trc=[64 98 103] , regionShape=[64 9 14], imageShape=[64 178 155] ]
```

We have specified an RA and DEC for the blc and the trc (they should be quantities; for blc we do that explicitly, but for the trc we just give a vector of strings which is automatically converted for us to a vector of quantities).

From the {\stff summary} listing you can see that RA and DEC correspond to pixel axes 3 and 2 respectively (don't be confused by the dual listing for the spectral axis) so that is why the {\stfaf pixelaxes} argument is set to [3,2]. If we had set blc/trc in DEC/RA order then we would have put {\stfaf pixelaxes=[1,2]}. For the unspecified frequency axis, all pixels are selected.

Example

```
- ia.open('bork')
- csys = ia.coordsys()
- csys.summary()

Name             Proj Shape Tile   Coord value at pixel    Coord incr Units
```

```
--------------------------------------------------------------------------------
Right Ascension   SIN   155   31  17:42:29.303    90.00 -1.000000e+00 arcsec
Declination       SIN   178   89 -28.59.18.600    90.00  1.000000e+00 arcsec

- rg.setcoordinates(cs)
T
- blc = "-10pix -28.59.18.6"
- trc = "10pix  -28.59.1.6"
- r1 = rg.wbox(blc=blc,trc=trc,absrel="relref abs")   # pixelaxes defaults to [0,1]
Using private CoordinateSystem from image "bork"
- ia.boundingbox(r1)
[blc=[80 90] , trc=[100 107] , regionShape=[21 18] , imageShape=[155 178] ]


In this example, we use pixel coordinates relative to the reference
pixel for the RA axis and absolute world coordinates for the DEC axis.
We also set the state of the \regionmanager\ with a Coordinate
System to use when making world regions.  You can see that when the
region was made, a message was issued reminding you that the internal
Coordinate System from the image {\sff bork} was being used.
```

Example

```
- ia.open('hcn')
- csys = ia.coordsys()
- csys.summary()

Name             Proj Shape Tile   Coord value at pixel    Coord incr Units
--------------------------------------------------------------------------------
Right Ascension   SIN   155   31  17:42:29.303    90.00 -1.000000e+00 arcsec
Declination       SIN   178   89 -28.59.18.600    90.00  1.000000e+00 arcsec
Frequency                64   16  1.413350e+09     1.00  1.968717e+04 Hz
Velocity                      1.378053e+02     1.00 -4.174021e+00 km/s
T
-
- blc = "1.414E9Hz"
- trc = "1.4145GHz"
- r = rg.wbox(blc=blc, trc=trc, pixelaxes=[2], csys=cs)
- ia.boundingbox(r)
[blc=[1 1 34] , trc=[155 178 59] , regionShape=[155 178 26] , imageShape=[155 178 64] ]
```

In this example we only specified a region for the frequency axis (note we used different units for the blc and trc). Therefore, on application, the region selected for the RA and DEC axes is automatically the full image.

---

regionmanager.wpolygon.html

regionmanager.wpolygon - Function

1.1.2 Create a world polygon region with quantities

Description

This function (short-hand name `wpoly`) creates a 2D world polygon region. The polygon is specified by an `x` and a `y` vector. These must be quantities of a vector (the world box function allows both quantities of vectors and vectors of quantities). This means that the units are common to all elements of each vector. Thus, `qa.quantity([1,2,3],'rad')` (a quantity of a vector) is different from `qa.quantity("1rad 2rad 3rad")` (a vector of quantities) although the information that they specify is the same.
You specify which pixel axes in the image the `x` and `y` vectors pertain to with the `pixelaxes` argument. If you don't, it defaults to [0,1]. This specification is an important part of world regions.
You can specify whether the `x` and `y` vector coordinates are given as absolute coordinates (`absrel='abs'`) or relative to the reference pixel (`absrel='relref'`) or relative to the center of the image (`absrel='relcen'`). This argument applies to both the axes of the polygon.
You must also specify the Coordinate System with the `csys` argument. The Coordinate System is encapsulated in a `coordinates tool` and can be recovered from an image with the coordsys function. You can also set a default Coordinate System in the Regionmanager with the setcoordinates function.
In the Regionmanager we have defined units 'pix' and 'frac'; these are then known to the quanta system. This means that you can effectively define a pixel box (except for the stride capability) as a world box with most of the advantages of world regions (can be used for compound regions). However, it is still not very portable to other images because the coordinates are pixel based, not world based.
Note that the need to deal with the `pixelaxes` and `csys` is hidden from you when using the gui interface of the Regionmanager.

Arguments

Inputs		
x	The x vector; a vector of quantities	
	allowed:	any
	Default:	variant
y	The y vector; vector of quantities	
	allowed:	any
	Default:	variant
		Quantity vector
pixelaxes	which pixel axes; vector of integers ..default -1 means [0,1]	
	allowed:	intArray
	Default:	-1
		[1,2]
csys	Coordinate System	
	allowed:	record
	Default:	Private Coordinate System
absrel	Absolute or relative coordinates; possibilities are 'abs', 'rel', 'relcen'	
	allowed:	string
	Default:	abs
		'abs'
comment	A comment stored with the region	
	allowed:	string
	Default:	

Returns
record

Example

```
ia.open('myim.im')
csys = ia.coordsys()
x = ["3pix", "6pix", "9pix", "6pix", "5pix","5pix","3pix"]
y = ["3pix","4pix","7pix","9pix","7pix","5pix","5pix"]
r1 = rg.wpolygon(x=x, y=y, pixelaxes=[0,1], csys=csys.torecord())
stats = ia.statistics(region=r1)
ia.done()
```

We applied the 2D polygon, defined in the XY plane with absolute pixel coordinates, to a 3D image. Therefore, the third (Z) axis was automatically extended to the whole image.

---

coordsys-Tool.html

1.1.3 coordsys - Tool

Operations on CoordinateSystems
Requires:

Synopsis

Description

Summary

A Coordsys `tool` is used to store and manipulate a Coordinate System (we will use the term 'Coordinate System' interchangeably with 'Coordsys `tool`'). A Coordinate System is a collection of coordinates, such as a direction coordinate (E.g. RA/DEC), or a spectral coordinate (e.g. an LSRK frequency).
The main job of the Coordsys `tool` is to convert between absolute pixel and world (physical) coordinates. It also supports relative pixel and world coordinates (relative to reference location).
A Coordinate System is generally associated with an image (manipulated via an Image `tool`) but can also exist in its own right. An image is basically just a regular lattice of pixels plus a Coordinate System describing the mapping from pixel coordinate to world (or physical) coordinate.
Each coordinate is associated with a number of axes. For example, a direction coordinate has two coupled axes; a longitude and a latitude. A spectral coordinate has one axis. A linear coordinate can have an arbitrary number of axes, but they are all uncoupled. The Coordinate System actually maintains two kinds of axes; pixel axes and world axes.
As well as the coordinates, there is some extra information stored in the Coordinate System. This consists of the telescope, the epoch (date of observation), and the highly influential observer's name. The telescope (i.e. position on earth) and epoch are important if you want to, say, change a spectral coordinate from LSRK to TOPO.
For general discussion about celestial coordinate systems, see the papers by Mark Calabretta and Eric Greisen. Background on the WCS system and relevant papers (including the papers published in

`A&A 2002, 1061-1075 and 1077-1122`

can be found here. Note that the actual system implemented originally in CASA was based on a 1996 draft of these papers. The final papers are being implemented while new version of the defining library become available.

Coordinate formatting

Many of the Coordsys `tool` functions use a world coordinate value as an argument. This world value can be formatted in many ways.
Some functions (e.g. toworld) have a function argument called `format` which takes a string. This controls the format in which the coordinate is output and hence possibly input into some other function.
Possibilities for `format` are :

- 'n' - means the world coordinate is given as a numeric vector (actually doubles). The units are implicitly those returned by function units.
- 'q' - means the world coordinate is given as a vector of quantities (value and unit) - see the quanta module. If there is only one axis (e.g. spectral coordinate), you will get a single quantum only.
- 'm' - means the world coordinate is given as a record of measures - see the measures module.

  The record consists of fields named `direction`, `spectral`, `stokes`, `linear`, and `tabular`, depending upon which coordinate types are present in the Coordinate System.

  The `direction` field holds a direction measure.

  The `spectral` field holds further subfields `frequency`, `radiovelocity`, `opticalvelocity`, `betavelocity`. The `frequency` subfield holds a frequency measure. The `radiovelocity` subfield holds a doppler measure using the radio velocity definition. The `opticalvelocity` subfield holds a doppler measure using the optical velocity definition. The `betavelocity` subfield holds a doppler measure using the true or beta velocity definition.

  The `stokes` field just holds a string giving the Stokes type (not a real measure).

  The `linear` and `tabular` fields hold a vector of quanta (not a real measure).
- 's' - means the the world coordinate is given as a vector of formatted strings

You can give a combination of one or more of the allowed letters when using the `format` argument. The coordinate is given as a record, with possible fields 'numeric', 'quantity', 'measure' and 'string' where each of these fields is given as described above.

There are functions torel and toabs used to inter-convert between absolute and relative world and pixel coordinates. These functions have an argument `isworld` whereby you can specify whether the coordinate is a pixel coordinate or a world coordinate. In general, you should not need to use this argument because any coordinate variable generated by Coordsys `tool` functions 'knows' whether it is absolute or relative, world or pixel. . However, you may be

inputting a coordinate variable which you have generated in some other way, and then you may need this.

Stokes Coordinates
Stokes axes don't fit very well into our Coordinate model since they are not interpolatable. The alternative to having a Stokes Coordinate is having a Stokes pixel type (like double, complex). Both have their good and bad points. We have chosen to use a Stokes Coordinate.
With the Stokes Coordinate, any absolute pixel coordinate value must be in the range of 1 to $nStokes$, where $nStokes$ is the number of Stokes types in the Coordinate.
We define relative world coordinates for a Stokes axis to be the same as absolute world coordinates (it makes no sense to think of a relative value $XY - XX$ say).
You can use the specialized functions stokes and setstokes to recover and set new Stokes values in the Stokes Coordinate.

World and Pixel axes
The Coordinate System maintains what it calls pixel and world axes. The pixel axis is associated with, say, the axes of a lattice of pixels. The world axes describe notional world axes, generally in the same order as the pixel axes. However, they may be different. Imagine that a 3-D image is collapsed along one axis. The resultant image has 2 pixel axes. However, we can maintain the world coordinate for the collapsed axis (so we know the coordinate value still). Thus we have three world axes and two pixel axes. It is also possible for the C++ programmer to reorder these pixel and world axes. However, this is strongly discouraged, and you should never actually encounter a situation where the pixel and world axes are in different orders, but you may encounter cases where the number of world and pixela axes is different.
For those of us (CASA programmers) writing robust scripts, we must account for these possibilities, although the user really shouldn't bother. Thus, the `pixel` and `world` vectors return the pixel and world axes of the found coordinate.
The functions referencevalue, increment, units, and names return their vectors in world axis order. However, function referencepixel returns in pixel axis order (and the world vectors might have more values than the referencepixel vector).

Overview of Coordsys tool functions

- **Get/set -** Functions to get and set various items within the Coordinate System are
  - referencepixel - get the reference pixel
  - setreferencepixel - set the reference pixel
  - referencevalue - get the reference value

- setreferencevalue - set the reference value
- setreferencelocation - Set reference pixel and value to these values
- increment - get axis increments
- setincrement - set axis increments
- lineartransform - get linear transform
- setlineartransform - set linear transform
- names - get axis names
- setnames - set axis names
- units - get axis units
- setunits - set axis units
- stokes - get Stokes values
- setdirection - set Direction coordinate values
- setstokes - set Stokes values
- setspectral - set Spectral coordinate tabular values
- settabular - set Tabular coordinate tabular values
- projection - get direction coordinate projection
- setprojection - set direction coordinate projection
- referencecode - get reference codes
- setreferencecode - set reference codes
- restfrequency - get the spectral coordinate rest frequency
- setrestfrequency - set the spectral coordinate rest frequency
- epoch - get the epoch of observation
- setepoch - set the epoch of observation
- telescope - get the telecope of the observation
- settelescope - set the telecope of the observation
- observer - get observer name
- setobserver - set observer name

- **Utility -** There is a range of utility services available through the functions

  - axesmap - get mapping between pixel and world axes order
  - axiscoordinatetypes - get type of coordinate for each axis
  - coordinatetype - get type of coordinates
  - copy - make a copy of this tool
  - done - destroy this `tool`

- findaxis - find specified axis (by number) in coordinate system
- findcoordinate - find specified (by number) coordinate
- fromrecord - set Coordinate System from a casapy record
- id - get the fundamental identifier of this `tool`
- naxes - get number of axes
- ncoordinates - get the number of coordinates
- reorder - reorder coordinates
- summary - summarize the Coordinate System
- torecord - Convert a Coordinate SYstem to a casapy record
- type - the type of this `tool`

- **Coordinate conversion**
  - convert - Convert one numeric coordinate with mixed input and output formats (abs/rel/world/pixel)
  - toabs - Convert a relative coordinate to an absolute coordinate
  - topixel - Convert from absolute world coordinate to absolute pixel coordinate
  - torel - Convert an absolute coordinate to a relative coordinate
  - toworld - Convert from an absolute pixel coordinate to an absolute world coordinate
  - convertmany - Convert many numeric coordinates with mixed input and output formats (abs/rel/world/pixel)
  - toabsmany - Convert many relative coordinates to absolute coordinates
  - topixelmany - Convert many absolute world coordinates to absolute pixel coordinates
  - torelmany - Convert many absolute coordinates to relative coordinates
  - toworldmany - Convert many absolute pixel coordinates to absolute world coordinates
  - frequencytovelocity - Convert from frequency to velocity
  - frequencytofrequency - Apply a relativistic Doppler shift to a list of frequencies
  - velocitytofrequency - Convert from velocity to frequency
  - setconversiontype - Set extra reference frame conversion layer
  - conversiontype - Recover extra reference frame conversion types

- **Tests -**

– coordsystest - Run test suite for Coordsys `tool`

Methods

newcoordsys	Create a non-default coordsys tool
addcoordinate	Add default coordinates. (For assay testing only.)
axesmap	Find mapping between world and pixel axes
axiscoordinatetypes	Return types of coordinates for each axis
conversiontype	Get extra reference conversion layer
convert	Convert a numeric mixed coordinate
convertdirection	Convert the direction coordinate to the specified frame by rotating as necessary abo
convertmany	Convert many numeric mixed coordinates
coordinatetype	Return type of specified coordinate
copy	Copy this Coordsys tool
done	Destroy this Coordsys tool, restore default tool
epoch	Return the epoch
findaxis	Find specified axis in coordinate system
findaxisbyname	Find specified axis in coordinate system.
findcoordinate	Find axes of specified coordinate
frequencytofrequency	Apply relativistic Doppler shift to a list of frequencies
frequencytovelocity	Convert frequency to velocity
fromrecord	Fill Coordinate System from a record
increment	Recover the increments
lineartransform	Recover the linear transform matrix
names	Recover the names for each axis
naxes	Recover the number of axes
ncoordinates	Recover the number of coordinates in the Coordinate System
observer	Return the name of the observer
projection	Recover the direction coordinate projection
referencecode	Return specified reference code
referencepixel	Recover the reference pixel
referencevalue	Recover the reference value
reorder	Reorder the coordinates
transpose	Transpose the axes.
replace	Replace a coordinate
restfrequency	Recover the rest frequency
setconversiontype	Set extra reference conversion layer
getconversiontype	Get extra reference conversion layer (aka conversiontype).
setdirection	Set direction coordinate values
setepoch	Set a new epoch
setincrement	Set the increment
setlineartransform	Set the linear transform
setnames	Set the axis names
setobserver	Set a new observer
setprojection	Set the direction coordinate projection
setreferencecode	Set new reference code
setreferencelocation	Set reference pixel and value
setreferencepixel	Set the reference pixel
setreferencevalue	Set the reference value
setrestfrequency	Set the rest frequency

setspectral	Set tabular values for the spectral coordinate
setstokes	Set the Stokes types
settabular	Set tabular values for the tabular coordinate
settelescope	Set a new telescope
setunits	Set the axis units
stokes	Recover the Stokes types
summary	Summarize basic information about the Coordinate System
telescope	Return the telescope
toabs	Convert relative coordinate to absolute
toabsmany	Convert many numeric relative coordinates to absolute
topixel	Convert from absolute world to pixel coordinate
topixelmany	Convert many absolute numeric world coordinates to pixel
torecord	Convert Coordinate System to a record
subimage	delivers a coordinate origin re-referenced for a subimage
torel	Convert absolute coordinate to relative
torelmany	Convert many numeric absolute coordinates to relative
toworld	Convert from absolute pixel coordinate to world
toworldmany	Convert many absolute pixel coordinates to numeric world
type	Return the type of this tool
units	Recover the units for each axis
velocitytofrequency	Convert velocity to frequency
parentname	Get parent image name.
setparentname	Set the parent image name (normally not needed by end-users)

coordsys.newcoordsys.html

coordsys.newcoordsys - Function

1.1.3 Create a non-default coordsys tool

Description

By default, this constructor makes an empty Coordsys `tool`. You can ask it to include various sorts of coordinates through the arguments. Except for Stokes, you don't have any control over the coordinate contents (e.g. reference value etc.) it does make for you on request. But you can edit the Coordinate System after creation if you wish.
If you wish to make a Stokes coordinate, then you assign `stokes` to a string (or a vector of strings) saying which Stokes you want. CASA allows rather a lot of potential Stokes types.
Probably most useful is some combination of the basic I, Q, U, V, XX, YY, XY, YX, RR, LL, RL, and LR.
However, a more esoteric choice is also possible: RX, RY, LX, LY, XR, XL, YR, YL (these are mixed linear and circular), PP, PQ, QP, QQ (general quasi-orthogonal correlation products) RCircular, LCircular, Linear (single dish polarization types).
You can also specify some polarization 'Stokes' types: Ptotal (Polarized intensity ($(Q^2+U^2+V^2)^{1/2}$), Plinear (Linearly Polarized intensity ($(Q^2+U^2)^{1/2}$), PFtotal (Polarization Fraction (Ptotal/I)), PFlinear (Linear Polarization Fraction (Plinear/I)), and Pangle (Linear Polarization Angle ($0.5\ arctan(U/Q)$ in radians)).
Probably you will find the more unusual types aren't fully supported throughout the system.
You can make a LinearCoordinate with as many uncoupled axes as you like. Thus, `linear=2` makes one LinearCoordinate with 2 axes (think of it like a DirectionCoordinate which also has 2 axes [but coupled in this case], a longitude and a latitude).
If you make a TabularCoordinate, it is linear to start with. You can change it to a non-linear one by providing a list of pixel and world values to function settabular.

Arguments

Inputs		
direction	Make a direction coordinate ?	
	allowed:	bool
	Default:	false
spectral	Make a spectral coordinate ?	
	allowed:	bool
	Default:	false
stokes	Make a Stokes coordinate with these Stokes	
	allowed:	stringArray
	Default:	I Q U V XX YY XY YX RR LL RL LR
linear	Make a linear coordinate with this many axes	
	allowed:	int
	Default:	0
tabular	Make a tabular coordinate	
	allowed:	bool
	Default:	false

Returns
coordsys

Example

```
"""
#
print "\t----\t newcoordsys Ex 1 \t----"
cs1=cs.newcoordsys()
print 'ncoordinates =',cs1.ncoordinates()
#0
cs1.done()
#True
cs2=cs.newcoordsys(direction=T,stokes=['I','V'])
print 'ncoordinates =',cs2.ncoordinates()
#2L
print cs2.coordinatetype()
#['Direction', 'Stokes']
cs2.summary()
#
"""
```

```
The second Coordinate System contains a direction coordinate
and a Stokes coordinate.   This means that there are three ‘axes’
associated with the 2 coordinates.
```

---

coordsys.addcoordinate.html

coordsys.addcoordinate - Function

1.1.3 Add default coordinates. (For assay testing only.)

Description

Add default coordinates of the specified types. This function allows multiple coordinates of the same type which are not well supported. Use only for assay tests.

Arguments

Inputs		
direction	Add a direction coordinate ?	
	allowed:	bool
	Default:	false
spectral	Add a spectral coordinate ?	
	allowed:	bool
	Default:	false
stokes	Add a Stokes coordinate with these Stokes	
	allowed:	stringArray
	Default:	I Q U V XX YY XY YX RR LL RL LR
linear	Add a linear coordinate with this many axes	
	allowed:	int
	Default:	0
tabular	Add a tabular coordinate	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
"""
#
print "\t----\t addcoordinate Ex 1 \t----"
mycs=cs.newcoordsys()
mycs.addcoordinate(direction=T)
mycs.done()
#
"""
```

---

coordsys.axesmap.html

coordsys.axesmap - Function

1.1.3 Find mapping between world and pixel axes

Description

This function returns a vector describing the mapping from pixel to world or world to pixel axes. It is not for general user use.
See the discussion about pixel and world axis ordering. Generally they will be in the same order.

Arguments

Inputs		
toworld	Map from pixel to world ?	
	allowed:	bool
	Default:	true

Returns
intArray

Example

```
"""
#
print "\t----\t axesmap Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
csys.axesmap(T);
#[1L, 2L, 3L]
csys.axesmap(F);
#[1L, 2L, 3L]
#
"""
```

coordsys.axiscoordinatetypes.html

coordsys.axiscoordinatetypes - Function

1.1.3 Return types of coordinates for each axis

Description

This function returns a vector string giving the coordinate type for each axis (world or pixel) in the Coordinate System.
See the discussion about pixel and world axis ordering.

Arguments

Inputs		
world	World or pixel axes ?	
	allowed:	bool
	Default:	true

Returns
stringArray

Example

```
"""
#
print "\t----\t axiscoordinatetypes Ex 1 \t----"
csys=cs.newcoordsys(direction=T,spectral=T)
csys.axiscoordinatetypes()
#['Direction', 'Direction', 'Spectral']
#
"""
```

coordsys.conversiontype.html

coordsys.conversiontype - Function

1.1.3 Get extra reference conversion layer

Description

Some coordinates contain a reference code. Examples of reference codes are B1950 and J2000 for direction coordinates, or LSRK and BARY for spectral coordinates. When you do conversions between pixel and world coordinate, the coordinates are in the reference frame corresponding to these codes. Function setconversiontype allows you to specify a different reference frame which is used when converting between world and pixel coordinate.
This function allows you to recover those conversion types. If no extra conversion layer has been set, you get back the native reference types.

Arguments

Inputs		
type	Coordinate type, direction, spectral	
	allowed:	string
	Default:	direction

Returns

string

Example

```
"""
#
print "\t----\t conversiontype Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
print csys.conversiontype (type='direction'), ' ', csys.conversiontype (type='spectral')
#J2000   LSRK
csys.setconversiontype (direction='GALACTIC', spectral='BARY')
print csys.conversiontype (type='direction'), ' ', csys.conversiontype (type='spectral')
```

```
#GALACTIC   BARY
#
"""
```

---

coordsys.convert.html

coordsys.convert - Function

1.1.3 Convert a numeric mixed coordinate

Description

This function converts between mixed pixel/world/abs/rel numeric coordinates. The input and output coordinates are specified via a numeric vector giving coordinate values, a string vector giving units, a boolean vector specifying whether the coordinate is absolute or relative (to the reference pixel) and doppler strings specifying the doppler convention for velocities. The units string may include `pix` for pixel coordinates and velocity units (i.e. any unit consistent with `m/s`).
The allowed doppler strings and definition are described in function summary.
The `shape` argument is optional. If your Coordinate System is from an image, then assign the image shape to this argument. It is used only when making mixed (pixel/world) conversions for Direction Coordinates to resolve ambiguity.
The example clarifies the use of this function.

Arguments

Inputs		
coordin	Input coordinate, as a numeric vector	
	allowed:	doubleArray
	Default:	
absin	Are input coordinate elements absolute ?	
	allowed:	boolArray
	Default:	true
dopplerin	Input doppler type for velocities	
	allowed:	string
	Default:	radio
unitsin	Input units, string vector	
	allowed:	stringArray
	Default:	Native
absout	Are output coordinate elements absolute ?	
	allowed:	boolArray
	Default:	true
dopplerout	Output doppler type for velocities	
	allowed:	string
	Default:	radio
unitsout	Output units	
	allowed:	stringArray
	Default:	Native
shape	Image shape, integer vector	
	allowed:	intArray
	Default:	-1

Returns
doubleArray

Example

```
In this example we convert from a vector of absolute pixels
to a mixture of pixel/world and abs/rel.

"""
#
print "\t----\t convert Ex 1 \t----"
csys=cs.newcoordsys(direction=T, spectral=T)    # 3 axes
cout=csys.convert(coordin=[10,20,30],absin=[T,T,T],
                  unitsin=["pix","pix","pix"],
                  absout=[T,F,T], dopplerout='optical',
```

```
                  unitsout=["pix","arcsec","km/s"])
print cout
#[10.0, 1140.0058038878046, 1139.1354056919731]
#
"""
```

---

coordsys.convertdirection.html

coordsys.convertdirection - Function

1.1.3 Convert the direction coordinate to the specified frame by rotating as necessary about the reference pixel so the axes line up with the cardinal directions.

Description

Convert the direction coordinate in the coordinate system to the specified frame by rotating about the reference pixel so that the resulting coordinate axes are parallel to the cardinal directions. The resulting coordinate will not have a conversion layer, even if the input direction coordinate does. A conversion layer can be set after by running cs.setconversiontype(). Be aware that if you attach the resulting coordinate system to an image whose pixels have not been rotated around the reference pixel in the same manner, you will likely get an image for which the pixels do not match up to world coordinate values. This method should only be used by experienced users who know what they are doing. It was written originally to facilitate rotating the direction coordinate since the implementation of imregrid requires this in certain circumstances. The conversion is done in place; a new coordinate system tool is not created. The returned record represents an angular quantity through which the old direction coordinate was rotated to create the new coordinate.

Arguments

Inputs		
frame	Reference frame to convert to.	
	allowed:	string
	Default:	

Returns

record

Example

coordsys.convertmany.html

coordsys.convertmany - Function

1.1.3 Convert many numeric mixed coordinates

Description

This function converts between many mixed pixel/world/abs/rel numeric coordinates. See function convert for more information.
The only diffference with that function is that you provide a matrix holding many coordinates to convert and a matrix of many converted coordinates is returned.

Arguments

Inputs		
coordin	Input coordinate, numeric matrix	
	allowed:	any
	Default:	variant
absin	Are input coordinate elements absolute ?	
	allowed:	boolArray
	Default:	true
dopplerin	Input doppler type for velocities	
	allowed:	string
	Default:	radio
unitsin	Input units, string vector	
	allowed:	stringArray
	Default:	Native
absout	Are output coordinate elements absolute ?	
	allowed:	boolArray
	Default:	true
dopplerout	Output doppler type for velocities	
	allowed:	string
	Default:	radio
unitsout	Output units	
	allowed:	stringArray
	Default:	Native
shape	Image shape, integer array	
	allowed:	intArray
	Default:	-1

Returns
variant

Example

```
"""
#
print "\t----\t convertmany Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)    # 3 axes
# absolute pixel coordinates; 10 conversions each of length 3; spectral
cin=[(15, 15, 15, 15, 15, 15, 15, 15, 15, 15),  # pixel runs from 1 to 10
     (20, 20, 20, 20, 20, 20, 20, 20, 20, 20),
     ( 1,  2,  3,  4,  5,  6,  7,  8,  9, 10)]
cout = csys.convertmany (coordin=cin,
                         absin=[T,T,T],
                         unitsin=["pix","pix","pix"],
                         absout=[T,F,T],
                         dopplerout='optical',
                         unitsout=["pix","deg","km/s"]);
print cout
#[(15.0, 15.0, 15.0, 15.0, 15.0, 15.0, 15.0, 15.0, 15.0, 15.0),
# (0.31666827885771637, 0.31666827885771637, 0.31666827885771637,
#  0.31666827885771637, 0.31666827885771637, 0.31666827885771637,
#  0.31666827885771637, 0.31666827885771637, 0.31666827885771637,
#  0.31666827885771637),
# (1145.3029083129913, 1145.0902316004676, 1144.8775551885467,
#  1144.6648790772279, 1144.4522032665102, 1144.2395277563601,
#  1144.0268525468437, 1143.8141776379266, 1143.6015030296085,
#  1143.3888287218554)]
#
"""
```

coordsys.coordinatetype.html

coordsys.coordinatetype - Function

1.1.3 Return type of specified coordinate

Description

This function returns a string describing the type of the specified coordinate.
If `which=unset` the types for all coordinates are returned.
Possible output values are 'Direction', 'Spectral', 'Stokes', 'Linear', and 'Tabular'

Arguments

Inputs		
which	Which coordinate ? (0-rel)	
	allowed:	int
	Default:	-1

Returns

stringArray

Example

```
"""
#
print "\t----\t coordinatetype Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
csys.coordinatetype(0)
#'Direction'
cs.coordinatetype()
#['Direction', 'Spectral']
#
"""
```

coordsys.copy.html

coordsys.copy - Function

1.1.3 Copy this Coordsys tool

Description

This function returns a copy, not a reference, of the Coordsys `tool`. It is your responsibility to call the `done` function on the new `tool`.

Arguments

Returns
coordsys

Example

```
"""
#
print "\t----\t copy Ex 1 \t----"
cs1 = cs.newcoordsys(direction=T, spectral=T)
cs2 = cs1         # Reference
print cs1, cs2
cs1.summary()
cs2.summary()
cs1.done()        # done invokes default coordsys tool
cs1.summary()
cs2.summary()     # cs2 gets doned when cs1 does
cs1 = cs.newcoordsys(direction=T, spectral=T)
cs2 = cs1.copy()  # Copy
cs1.done()
cs1.summary()     # cs1 is default coordsys tool
cs2.summary()     # cs2 is still viable
```

```
cs2.done()
cs2.summary()     # Now it's done (done just invokes default constructor)
#
"""
```

---

coordsys.done.html

coordsys.done - Function

1.1.3 Destroy this Coordsys tool, restore default tool

Description

If you no longer need to use a Coordsys `tool`calling this function will free up its resources and restore the default coordsys tool.

Arguments

Returns
bool

Example

```
"""
#
print "\t----\t done Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
csys.done()
print csys.torecord()            # default tool
#
"""
```

---

coordsys.epoch.html

coordsys.epoch - Function

1.1.3 Return the epoch

Description

This function returns the epoch of the observation as a Measure.

Arguments

Returns
record

Example

```
"""
#
print "\t----\t epoch Ex 1 \t----"
csys = cs.newcoordsys()
ep = csys.epoch()
print ep
#{'type': 'epoch', 'm0': {'value': 54151.96481085648, 'unit': 'd'}, 'refer': 'UTC'}
time = me.getvalue(ep)           # Extract time with measures
print time
#{'m0': {'value': 54151.96481085648, 'unit': 'd'}}
qa.time(time)                    # Format with quanta
#'23:09:20'
#
"""
```

coordsys.findaxis.html

coordsys.findaxis - Function

1.1.3 Find specified axis in coordinate system

Description

This function finds the specified axis in the Coordinate System. If the axis does not exist, it throws an exception.

Arguments

Inputs		
world	is axis a world or pixel axis ?	
	allowed:	bool
	Default:	true
axis	Axis in coordinate system	
	allowed:	int
	Default:	0

Returns

record

Example

```
"""
#
print "\t----\t findaxis Ex 1 \t----"
csys=cs.newcoordsys(direction=T, linear=2)          # RA/DEC/Lin1/Lin2
rtn=csys.findaxis(T,1)                              # DEC
rtn
#{'axisincoordinate': 1L, 'coordinate': 0L}
rtn = csys.findaxis(T,2)                            # Lin1
rtn
#{'axisincoordinate': 0L, 'coordinate': 1L}
#
```

```
"""

In these examples, the Coordinate System has 2 coordinates and 4 axes
(0-rel, both world and pixel the same).  The first example finds the
DEC axis (coordinate system axis 1) to be the second axis of the
Direction Coordinate (coordinate 0). The second example finds the
first linear axis (coordinate system axis 2) to be the first axis of
the Linear Coordinate (coordinate 1).
```

---

coordsys.findaxisbyname.html

coordsys.findaxisbyname - Function

1.1.3 Find specified axis in coordinate system.

Description

Find the world axis based on its name. Matching is not case sensitive and minimal match is supported, eg "dec" will match "Declination". In addition, if allowfriendlyname is True, other common terms will match the expected axis. Currently supported are: "spectral" matches frequency type axes, eg "Frequency" or "Velocity", "ra" matches "Right Ascension". These names must be spelled out completely; eg "spectral" rather than simply "spec". The first matching axis (zero-based) number is returned. If no axis can be matched, an exception is thrown.

Arguments

Inputs	
axisname	Name of axis to find. Minimal match supported allowed: string Default:
allowfriendlyname	Support friendly naming. Eg "spectral" will match "frequency" or "velocity", "ra" will match "right ascension" allowed: bool Default: true

Returns

int

Example

```
# Find the declination axis
ia.open("myimage")
csys = ia.coordsys()
ia.done()
try:
```

```
    dec_axis_number = csys.findaxisbyname("dec", False)
except Exception
    print "Declination axis not found

# find the spectral axis
try:
    spec_axis_number = csys.findaxisbyname("spectral", True)
except Exception:
    print "Spectral axis could not be found.
```

---

coordsys.findcoordinate.html

coordsys.findcoordinate - Function

1.1.3 Find axes of specified coordinate

Description

This function finds the axes in the Coordinate System for the specified coordinate (minimum match is active for argument `type`). By default it finds the first coordinate, but if there is more than one (can happen for linear coordinates), you can specify which. It returns a dictionary with 'return', 'pixel', and 'world' as keys. The associated value of 'return' is a boolean indicating if the specified coordinate was found. The values of 'pixel' and 'world' are arrays indicating the indices of the associated pixel and world axes, respectively, of the specified coordinate. If the coordinate does not exist, these arrays will be empty.
See also the function axesmap which returns the mapping between pixel and world axes.

Arguments

Inputs		
type	Type of coordinate to find: direction, stokes, spectral, linear, or tabular	
	allowed:	string
	Default:	direction
which	Which coordinate if more than one	
	allowed:	int
	Default:	0

Returns
record

Example

"""

```
#
print "\t----\t findcoordinate Ex 1 \t----"
csys=cs.newcoordsys(direction=T)
rtn=cs.findcoordinate('direction')
print rtn
#{'world': [0L, 1L], 'pixel': [0L, 1L]}
print 'pixel, world axes =', rtn['pixel'], rtn['world']
#pixel, world axes = [0 1] [0 1]
#
"""
```

---

coordsys.frequencytofrequency.html

coordsys.frequencytofrequency - Function

1.1.3 Apply relativistic Doppler shift to a list of frequencies

Description

This function converts frequencies to frequencies by applying a relativistic Doppler shift: fout = fin * sqrt((1.-v/c)/(1.+v/c)) .
The input frequencies are specified via a vector of numeric values and a specified unit (`frequnit`). If you don't give a frequency unit, it is assumed that the units are those given by function coordsys units() for the spectral coordinate.
This function does not make any frame conversions (e.g. LSR to BARY).
This function fails if there is no spectral coordinate in the Coordinate System.
See also function frequencytovelocity.

Arguments

Inputs		
value	Frequencies to convert	
	allowed:	doubleArray
	Default:	
frequnit	Unit of input frequencies. Default is unit of the spectral coordinate.	
	allowed:	string
	Default:	
velocity	Velocity	
	allowed:	any
	Default:	variant

Returns
doubleArray

Example

```
"""
  ia.open('M100line.image')
  mycs = ia.coordsys()
  ia.close()

  mycs.frequencytofrequency(value=[115271201800.0], frequnit='Hz', velocity='1000km/s')
results in
  array([114887337607.0])

Let's see if this is correct
  print 115271201800.0*sqrt((1.-1000000./299792458.0)/(1.+1000000./299792458.0))
Result: 1.14887337607e+11
"""
```

---

coordsys.frequencytovelocity.html

coordsys.frequencytovelocity - Function

1.1.3 Convert frequency to velocity

Description

This function converts frequencies to velocities.
The input frequencies are specified via a vector of numeric values and a specified unit (`frequnit`). If you don't give a frequency unit, it is assumed that the units are those given by function coordsys units() for the spectral coordinate.
This function does not make any frame conversions (e.g. LSR to BARY) but you can specifiy the velocity doppler definition via the `doppler` argument (see image summary() for possible values).
The velocities are returned in a vector for which you specify the units (`velunit` - default is km/s).
This function will return a fail if there is no spectral coordinate in the Coordinate System. See also function velocitytofrequency.

Arguments

Inputs		
value	Frequency to convert	
	allowed:	doubleArray
	Default:	
frequnit	Unit of input frequencies. Default is unit of the spectral coordinate.	
	allowed:	string
	Default:	
doppler	Velocity doppler definition	
	allowed:	string
	Default:	radio
velunit	Unit of output velocities	
	allowed:	string
	Default:	km/s

Returns
doubleArray

Example

```
"""
#
print "\t----\t frequencytovelocity Ex 1 \t----"
im = ia.fromshape(shape=[10,10,10])
csys = ia.coordsys()
rtn = csys.findcoordinate('spectral')   # Find spectral axis
pa=rtn['pixel']
wa=rtn['world']
pixel = csys.referencepixel();          # Use reference pixel for non-spectral
nFreq = ia.shape()[pa];                 # Length of spectral axis
freq = [];
for i in range(nFreq):
  pixel[pa] = i            # Assign value for spectral axis of pixel coordinate
  w = csys.toworld(value=pixel, format='n')   # Convert pixel to world
  freq.append(w['numeric'][wa]);              # Fish out frequency
print "freq=", freq
#freq= [1414995000.0, 1414996000.0, 1414997000.0, 1414998000.0,
# 1414999000.0, 1415000000.0, 1415001000.0, 1415002000.0, 1415003000.0, 1415004000.0]
vel = csys.frequencytovelocity(value=freq, doppler='optical', velunit='km/s')
print "vel=", vel
#vel= [1146.3662963847394, 1146.153618169159, 1145.9409402542183, 1145.7282626398826,
# 1145.5155853261515, 1145.3029083129911, 1145.0902316004676, 1144.8775551885467,
# 1144.6648790772279, 1144.4522032665104]
#
"""
```

In this example, we find the optical velocity in km/s of every pixel along the spectral axis of our image. First we obtain the Coordinate System from the image. Then we find which axis of the Coordinate System (image) pertain to the spectral coordinate. Then we loop over each pixel of the spectral axis, and convert a pixel coordinate (one for each axis of the image) to world. We obtain the value for the spectral axis from that world vector, and add it to the vector of frequencies. Then we convert that vector of frequencies to velocity.

---

coordsys.fromrecord.html

coordsys.fromrecord - Function

1.1.3 Fill Coordinate System from a record

Description

You can convert a Coordinate System to a record (torecord). This function (fromrecord) allows you to set the contents of an existing Coordinate System from such a record. In doing so, you overwrite its current contents.

Arguments

Inputs		
record	Record containing Coordinate System	
	allowed:	record
	Default:	

Returns

bool

Example

```
"""
#
print "\t----\t fromrecord Ex 1 \t----"
csys = cs.newcoordsys(direction=T, stokes="I Q")
print csys.ncoordinates()
#2
r = csys.torecord()
cs2 = cs.newcoordsys()
print cs2.ncoordinates()
#0
cs2.fromrecord(r)
print cs2.ncoordinates()
#2
```

```
#
"""
```

---

coordsys.increment.html

coordsys.increment - Function

1.1.3 Recover the increments

Description

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.
This function returns the increment (in world axis order). You can recover the increments either for all coordinates (leave `type` unset) or for a specific coordinate type (mimumum match of the allowed types will do). If you ask for a non-existent coordinate an exception is generated.
See the discussion regarding the formatting possibilities available via argument `format`.
You can set the increment with function setincrement.

Arguments

Inputs		
format	Format string from combination of ”n”, ”q”, ”s”, ”m”	
	allowed:	string
	Default:	n
type	Coordinate type: ”direction”, ”stokes”, ”spectral”, ”linear”, ”tabular”. Leave empty for all.	
	allowed:	string
	Default:	

Returns
record

Example

```
"""
#
```

```
print "\t----\t increment Ex 1 \t----"
csys=cs.newcoordsys(direction=T,spectral=T)
print csys.increment(format='q')
#{'quantity': {'*1': {'unit': "'", 'value': -1.0},
#              '*2': {'unit': "'", 'value': 1.0},
#              '*3': {'unit': 'Hz', 'value': 1000.0}}}
print csys.increment(format='n')
#{'numeric': [-1.0, 1.0, 1000.0]}
print csys.increment(format='n', type='spectral')
#{'numeric': [1000.0]}
#
"""
```

---

coordsys.lineartransform.html

coordsys.lineartransform - Function

1.1.3 Recover the linear transform matrix

Description

Recover the linear transform component for the specified coordinate type. You can set the linear transform with function setlineartransform.

Arguments

Inputs		
type	Coordinate type: ”direction”, ”stokes”, ”spectral”, ”linear”, ”tabular”	
	allowed:	string
	Default:	

Returns

variant

Example

```
"""
#
print "\t----\t lineartransform Ex 1 \t----"
csys=cs.newcoordsys(direction=T,linear=3)
csys.lineartransform('dir')                              # 2 x 2
# [(1.0, 0.0), (0.0, 1.0)]
csys.lineartransform('lin')                              # 3 x 3
# [(1.0, 0.0, 0.0), (0.0, 1.0, 0.0), (0.0, 0.0, 1.0)]
#
"""
```

coordsys.names.html

coordsys.names - Function

1.1.3 Recover the names for each axis

Description

Each axis associated with the Coordinate System has a name (they don't mean anything fundamental). This function returns those names in world axis order. You can recover the names either for all coordinates (leave `type` unset) or for a specific coordinate type (mimumum match of the allowed types will do). If you ask for a non-existent coordinate an exception is generated.
You can set the names with function setnames.

Arguments

Inputs		
type	Coordinate type: "direction", "stokes", "spectral", "linear", "tabular". Leave empty for all.	
	allowed:	string
	Default:	

Returns
stringArray

Example

```
"""
#
print "\t----\t names Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
n = csys.names()
print n[0]
#Right Ascension
print n[1]
#Declination
```

```
print n[2]
#Frequency
print cs.names('spec')
#Frequency
#
"""
```

---

coordsys.naxes.html

coordsys.naxes - Function

1.1.3 Recover the number of axes

Description

Find the number of axes in the Coordinate System.
You may find the number of world or pixel axes; these are generally the same and general users can ignore the distinction. See the discussion about pixel and world axis ordering.

Arguments

Inputs		
world	Find number of world or pixel axes ?	
	allowed:	bool
	Default:	true

Returns
int

Example

```
"""
#
print "\t----\t naxes Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
n = csys.naxes(T)
print n
#3                          # 2 direction axes, 1 spectral
n = csys.naxes(F)
print n
#3
#
"""
```

coordsys.ncoordinates.html

coordsys.ncoordinates - Function

1.1.3 Recover the number of coordinates in the Coordinate System

Description

This function recovers the number of coordinates in the Coordinate System.

Arguments

Returns
int

Example

```
"""
#
print "\t----\t ncoordinates Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
print csys.ncoordinates()
#2
cs2 = cs.newcoordsys(linear=4)
print cs2.ncoordinates()
#1
#
"""
```

---

coordsys.observer.html

coordsys.observer - Function

1.1.3 Return the name of the observer

Description

This function returns the name of the observer. You can set it with the function setobserver.

Arguments

Returns
string

Example

```
"""
#
print "\t----\t observer Ex 1 \t----"
csys = cs.newcoordsys()
print csys.observer()
#Karl Jansky
#
"""
```

---

coordsys.projection.html

coordsys.projection - Function

1.1.3 Recover the direction coordinate projection

Description

If the Coordinate System contains a direction coordinate, this function can be used to recover information about the projection. For discussion about celestial coordinate systems, including projections, see the papers by Mark Calabretta and Eric Greisen. The initial draft from 1996 (implemented in CASA. Background information can be found here.
What this function returns depends upon the value you assign to `type`.

- `type=unset`. In this case (the default), the actual projection type and projection parameters are returned in a record with fields `type` and `parameters`, respectively.
- `type='all'`. In this case, a vector of strings containing all of the possible projection codes is returned.
- `type=code`. If you specify a valid projection type code (see list by setting `type='all'`) then what is returned is the number of parameters required to describe that projection (useful in function setprojection).

You can change the projection with setprojection.
If the Coordinate System does not contain a direction coordinate, an exception is generated.

Arguments

Inputs		
type	Type of projection. Defaults to current projection.	
	allowed:	string
	Default:	

Returns
record

Example

```
"""
#
print "\t----\t projection Ex 1 \t----"
csys = cs.newcoordsys(direction=T)
print csys.projection()
#{'type': 'SIN', 'parameters': [0.0, 0.0]}
print csys.projection('all')
#{'all': True, 'types': ['AZP', 'TAN', 'SIN', 'STG', 'ARC', 'ZPN', 'ZEA',
# 'AIR', 'CYP', 'CAR', 'MER', 'CEA', 'COP', 'COD', 'COE', 'COO', 'BON',
# 'PCO', 'SFL', 'PAR', 'AIT', 'MOL', 'CSC', 'QSC', 'TSC']}
print csys.projection('ZPN')
#{'nparameters': 100}
#
"""
```

We first recover the projection type and parameters from the direction coordinate. Then we find the list of all possible projection types. FInally, we recover the number of parameters required to describe the 'ZPN' projection.

---

coordsys.referencecode.html

coordsys.referencecode - Function

1.1.3 Return specified reference code

Description

This function returns the reference code for all, or the specified coordinate type. Examples of the reference code are B1950 and J2000 for direction coordinates, or LSRK and BARY for spectral coordinates.
If `type` is left unset, then a vector of strings is returned, one code for each coordinate type in the Coordinate System.
If you specify `type` then select from 'direction', 'spectral', 'stokes', and 'linear' (the first two letters will do). However, only the first two coordinate types will return a non-empty string. If the Coordinate System does not contain a coordinate of the type you specify, an exception is generated.
The argument `list` is ignored unless you specify a specific `type`. If `list=T`, then this function returns the list of all possible reference codes for the specified coordinate type. Otherwise, it just returns the actual code current set in the Coordinate System.
The list of all possible types is returned as a record (it is actually generated by the listcodes function in the measures system). This record has two fields. These are called 'normal' (containing all normal codes) and 'extra' (maybe empty, with all extra codes like planets).
You can set the reference code with setreferencecode.

Arguments

Inputs		
type	Coordinate type: "direction", "stokes", "spectral", "linear", "tabular". Leave empty for all.	
	allowed:	string
	Default:	
list	List all possibilities?	
	allowed:	bool
	Default:	false

Returns

stringArray

Example

```
"""
#
print "\t----\t referencecode Ex 1 \t----"
csys = cs.newcoordsys(direction=T)
clist = csys.referencecode('dir', T)
print clist
#['J2000', 'JMEAN', 'JTRUE', 'APP', 'B1950', 'BMEAN', 'BTRUE',
# 'GALACTIC', 'HADEC', 'AZEL', 'AZELSW', 'AZELNE', 'AZELGEO',
# 'AZELSWGEO', 'AZELNEGEO', 'JNAT', 'ECLIPTIC', 'MECLIPTIC',
# 'TECLIPTIC', 'SUPERGAL', 'ITRF', 'TOPO', 'ICRS',
# 'MERCURY', 'VENUS', 'MARS', 'JUPITER', 'SATURN', 'URANUS',
# 'NEPTUNE', 'PLUTO', 'SUN', 'MOON', 'COMET']
print csys.referencecode('dir')
#J2000
#
"""
```

In this example we first get the list of all possible reference codes ofor a direction coordinate. Then we
get the actual reference code for the direction coordinate in our Coordinate System.

---

coordsys.referencepixel.html

coordsys.referencepixel - Function

1.1.3 Recover the reference pixel

Description

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.
This function returns the reference pixel (in pixel axis order). You can recover the reference pixel either for all coordinates (leave `type` unset) or for a specific coordinate type (mimumum match of the allowed types will do). If you ask for a non-existent coordinate an exception is generated.
You can set the reference pixel with function setreferencepixel.

Arguments

Inputs	
type	Coordinate type: "direction", "stokes", "spectral", "linear", "tabular". Leave empty for all. allowed: string Default:

Returns
record

Example

```
"""
#
print "\t----\t referencepixel Ex 1 \t----"
csys = cs.newcoordsys(spectral=T, linear=2)
csys.setreferencepixel([1.0, 2.0, 3.0])
print csys.referencepixel()
#{'ar_type': 'absolute', 'pw_type': 'pixel', 'numeric': array([ 1.,  2.,  3.])}
```

```
print csys.referencepixel('lin')
#{'ar_type': 'absolute', 'pw_type': 'pixel', 'numeric': array([ 2.,  3.])}
#
"""
```

---

coordsys.referencevalue.html

coordsys.referencevalue - Function

1.1.3 Recover the reference value

Description

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.
This function returns the reference value (in world axis order). You can recover the reference value either for all coordinates (leave `type` unset) or for a specific coordinate type (mimumum match of the allowed types will do). If you ask for a non-existent coordinate an exception is generated.
See the discussion regarding the formatting possibilities available via argument `format`.
You can set the reference value with function setreferencevalue.

Arguments

Inputs		
format	Format string. Combination of "n", "q", "s", "m"	
	allowed:	string
	Default:	n
type	Coordinate type: "direction", "stokes", "spectral", "linear", "tabular". Leave empty for all.	
	allowed:	string
	Default:	

Returns
record

Example

```
"""
#
```

```
print "\t----\t referencevalue Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
print csys.referencevalue(format='q')
#{'ar_type': 'absolute',
# 'pw_type': 'world',
# 'quantity': {'*1': {'unit': "'", 'value': 0.0},
#              '*2': {'unit': "'", 'value': 0.0},
#              '*3': {'unit': 'Hz', 'value': 1415000000.0}}}
print csys.referencevalue(format='n')
#{'ar_type': 'absolute',
# 'numeric': array([  0.00000000e+00,   0.00000000e+00,   1.41500000e+09]),
# 'pw_type': 'world'}
print csys.referencevalue(format='n', type='spec')
#{'ar_type': 'absolute',
# 'numeric': array([  1.41500000e+09]),
# 'pw_type': 'world'}
#
"""
```

---

coordsys.reorder.html

coordsys.reorder - Function

1.1.3 Reorder the coordinates

Description

This function reorders the coordinates in the Coordinate System. You specify the new order of the coordinates in terms of their old order.

Arguments

Inputs		
order	New coordinate order	
	allowed:	intArray
	Default:	

Returns

bool

Example

```
"""
#
print "\t----\t reorder Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T, linear=2)
print csys.coordinatetype()
#['Direction', 'Spectral', 'Linear']
csys.reorder([1,2,0]);
print csys.coordinatetype()
#['Spectral', 'Linear', 'Direction']
#
"""
```

coordsys.transpose.html

coordsys.transpose - Function

1.1.3 Transpose the axes.

Description

This method transposes the axes (both world and pixel) in the coordinate system. You specify the new order of the axes in terms of their old order, so eg order=[1,0,3,2] means reorder the axes so that the zeroth axis becomes the first axis, the first axis becomes the zeroth axis, the second axis becomes the third axis, and the third axis becomes the second axis.

Arguments

Inputs		
order	New axis order	
	allowed:	intArray
	Default:	

Returns

bool

Example

```
csys = cstool()

# Create a coordinate system with axes, RA, Dec, Stokes, and Frequency
csys.newcoordsys(direction=T, spectral=T, stokes=["I","Q"])

# transpose the axes so that the order is RA, Dec, Frequency, and Stokes
csys.transpose(order=[0, 1, 3, 2])
```

coordsys.replace.html

coordsys.replace - Function

1.1.3 Replace a coordinate

Description

This function replaces one coordinate in the current Coordinate System by one coordinate in the given Coordinate System. The specified coordinates must have the same number of axes.

Arguments

Inputs		
csys	Coordinate System to replace from. Use coordsys' torecord() to generate required record. allowed: record Default:	
whichin	Index of input coordinate (0-rel) allowed: int Default:	
whichout	Index of output coordinate allowed: int Default:	

Returns
bool

Example

```
"""
#
print "\t----\t replace Ex 1 \t----"
cs1 = cs.newcoordsys(direction=T, linear=1)
print cs1.coordinatetype()
#['Direction', 'Linear']
```

```
cs2 = cs.newcoordsys(spectral=T)
cs1.replace (cs2.torecord(),0,1)
print cs1.coordinatetype()
#['Direction', 'Spectral']
#
"""
```

---

coordsys.restfrequency.html

coordsys.restfrequency - Function

1.1.3 Recover the rest frequency

Description

If the Coordinate System contains a spectral coordinate, then it has a rest frequency. In fact, the spectral coordinate can hold several rest frequencies (to handle for example, an observation where the band covers many lines), although only one is active (for velocity conversions) at a time.
This function recovers the rest frequencies as a quantity vector. The first frequency is the active one.
You can change the rest frequencies with setrestfrequency.
If the Coordinate System does not contain a frequency coordinate, an exception is generated.

Arguments

Returns
record

Example

```
"""
#
print "\t----\t restfrequency Ex 1 \t----"
csys = cs.newcoordsys(spectral=T)
print csys.restfrequency()
#{'value': [1420405751.7860003], 'unit': 'Hz'}
csys.setrestfrequency (value=qa.quantity([1.2e9, 1.3e9],'Hz'), which=1, append=F)
print csys.restfrequency()
#{'value': [1300000000.0, 1200000000.0], 'unit': 'Hz'}
```

```
#
"""

In the example, the initial spectral coordinate has 1 rest frequency.
Then we set it with two, nominating the second as the active rest frequency,
and recover them.
```

---

coordsys.setconversiontype.html

coordsys.setconversiontype - Function

1.1.3 Set extra reference conversion layer

Description

Some coordinates contain a reference code. Examples of reference codes are B1950 and J2000 for direction coordinates, or LSRK and BARY for spectral coordinates. When you do conversions between pixel and world coordinate, the coordinates are in the reference frame corresponding to these codes.
This function allows you to specify a different reference frame which is used when converting between world and pixel coordinate (see function conversiontype to recover the conversion types). If it returns F, it means that although the conversion machines were successfully created, a trial conversion failed. This usually means the REST frame was involved which requires a radial velocity (not yet implemented). If this happens, the conversion type will be left as it was. The function fails if more blatant things are wrong like a missing coordinate, or an incorrect reference code.
The list of possible reference codes can be obtained via function referencecode. With this function, you specify the desired reference code. Then, when a conversion between pixel and world is requested, an extra conversion is done to (`toWorld`) or from (`toPixel`) the specified reference frame.
The summary function shows the extra conversion reference system to the right of the native reference system (if it is different) and in parentheses.
Note that to convert between different spectral reference frames, you need a position, epoch and direction. The position (telescope) and epoch (date of observation), if not in your coordinate system can be set with functions settelescope and setepoch. The direction is the reference direction of the *required* direction coordinate in the coordinate system.

As an example, let us say you are working with a spectral coordinate which was constructed with the LSRK reference frame. You want to convert some pixel coordinates to barycentric velocities (reference code BARY).

```
"""
#
print "\t----\t setconversiontype Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T); # Create coordinate system
rtn=csys.findcoordinate('spectral')             # Find spectral coordinate
wa=rtn['world']
pa=rtn['pixel']
```

```
u = csys.units()[wa]                              # Spectral unit
print csys.referencecode(type='spectral')         # Which is  in LSRK reference frame
#LSRK
p = [10,20,30]
w = csys.toworld(p, format='n')           # Convert a pixel to LSRK world
print 'pixel, world = ', p, w['numeric']
#pixel, world =  [10, 20, 30] [21589.999816660376, 20.000112822985134, 1415030000.0]
p2 = csys.topixel(w)                      # and back to pixel
print 'world, pixel = ', w['numeric'], p2
#world, pixel =  [21589.999816660376, 20.000112822985134, 1415030000.0]
# [10.00000000000248, 19.999999999999801, 30.0]
# Convert LSRK frequency to LSRK velocity
v = csys.frequencytovelocity(value=w['numeric'][wa], frequnit=u,
                             doppler='RADIO', velunit='m/s');
print 'pixel, frequency, velocity = ', p[pa], w['numeric'][wa], v
#pixel, frequency, velocity =  30 1415030000.0 1134612.30321
csys.setconversiontype(spectral='BARY')   # Specify BARY reference code
w = csys.toworld(p, format='n')           # Convert a pixel to BARY world
print 'pixel, world = ', p, w['numeric']
#pixel, world =  [10, 20, 30] [21589.999816660376, 20.000112822985134, 1415031369.0081882]
p2 = csys.topixel(w)                      # and back to pixel
print 'world, pixel = ', w['numeric'], p2
#world, pixel =  [21589.999816660376, 20.000112822985134, 1415031369.0081882]
# [10.00000000000248, 19.999999999999801, 30.0]
# Convert BARY frequency to BARY velocity
v = csys.frequencytovelocity(value=w['numeric'][wa], frequnit=u,
                               doppler='RADIO', velunit='m/s');
print 'pixel, frequency, velocity = ', p[pa], w['numeric'][wa], v
#pixel, frequency, velocity =  30 1415031369.01 1134323.35878
#
"""
```

You must also be aware of when this extra layer is active and when it is not. It's a bit nasty.

- - Whenever you use `toWorld`, `toPixel` `toWorldMany`, or `toPixelMany` the layer is active.
- - Whenever you use `convert` or `convertMany` the layer *may* be active. Here are the rules !

  It is only relevant to spectral and direction coordinates.

  For the direction coordinate part of your conversion, if you request a pure world or pixel conversion it is active. Any pixel/world mix will not invoke it (because it is ill defined).

  For the spectral coordinate part it is always active (only one axis so must be pixel or world).

- - This layer is irrelevant to all functions converting between frequency and velocity, and absolute and relative. The values are in whatever frame you are working with.

The summary function lists the reference frame for direction and spectral coordinates. If you have also set a conversion reference code it also lists that (to the right in parentheses).

Arguments

Inputs		
direction	Reference code	
	allowed:	string
	Default:	
spectral	Reference code	
	allowed:	string
	Default:	

Returns
bool

---

coordsys.getconversiontype.html

coordsys.getconversiontype - Function

1.1.3 Get extra reference conversion layer (aka conversiontype).

Description

See conversiontype for more complete description.

Arguments

Inputs		
type	Conversion type	
	allowed:	string
	Default:	
showconversion	Show the conversion layer	
	allowed:	bool
	Default:	true

Returns

string

---

coordsys.setdirection.html

coordsys.setdirection - Function

1.1.3 Set direction coordinate values

Description

When you construct a Coordsys `tool`, if you include a Direction Coordinate, it will have some default parameters. This function simply allows you to replace the values of the Direction Coordinate.
You can also change almost all of those parameters (such as projection, reference value etc.) via the individual functions setreferencecode, setprojection, setreferencepixel, setreferencevalue, setincrement, and setlineartransform provided by the Coordsys `tool`. See those functions for more details about the formatting of the above function arguments.
Bear in mind, that if your Coordinate System came from a real image, then the reference pixel is special and you should not change it.

Arguments

Inputs		
refcode	Reference code. Default is no change.	
	allowed:	string
	Default:	
proj	Projection type. Default is no change.	
	allowed:	string
	Default:	
projpar	Projection parameters. Default is no change.	
	allowed:	doubleArray
	Default:	-1
refpix	Reference pixel. Default is no change.	
	allowed:	doubleArray
	Default:	-1
refval	Reference value. Default is no change.	
	allowed:	variant
	Default:	
incr	Increment. Default is no change.	
	allowed:	variant
	Default:	
xform	Linear transform. Default is no change.	
	allowed:	variant
	Default:	
poles	Native poles. Default is no change.	
	allowed:	variant
	Default:	

Returns
bool

Example

```
"""
#
print "\t----\t setdirection Ex 1 \t----"
csys = cs.newcoordsys(direction=T);
csys.setdirection (refcode='GALACTIC', proj='SIN', projpar=[0,0],
                   refpix=[-10,20], refval="10deg -20deg");
print csys.projection()
#{'type': 'SIN', 'parameters': array([ 0.,  0.])}
print csys.referencepixel()
```

```
#{'ar_type': 'absolute', 'pw_type': 'pixel', 'numeric': array([-10.,  20.])}
print csys.referencevalue(format='s')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'string': array(['10.00000000 deg', '-20.00000000 deg'], dtype='|S17')}
#
"""
```

---

coordsys.setepoch.html

coordsys.setepoch - Function

1.1.3 Set a new epoch

Description

This function sets a new epoch (supplied as an epoch measure) of the observation. You can get the current epoch with function epoch.

Arguments

Inputs		
value	New epoch measure	
	allowed:	record
	Default:	

Returns

bool

Example

```
"""
#
print "\t----\t setepoch Ex 1 \t----"
csys = cs.newcoordsys()
ep = csys.epoch()
print ep
#{'type': 'epoch', 'm0': {'value': 54161.766782997685, 'unit': 'd'}, 'refer': 'UTC'}
ep = me.epoch('UTC', 'today')
csys.setepoch(ep)
print csys.epoch()
#{'type': 'epoch', 'm0': {'value': 54161.766782997685, 'unit': 'd'}, 'refer': 'UTC'}
#
"""
```

coordsys.setincrement.html

coordsys.setincrement - Function

1.1.3 Set the increment

Description

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.
This function allows you to set a new increment. You should not do this on "stokes" axes unless you are an adept or a big risk taker.
You can set the increments either for all axes (`type=unset`) or for just the axes associated with a particular coordinate type.
You may supply the increments in all of the formats described in the formatting discussion.
In addition, you can also supply the increments as a quantity of vector of doubles. For example `qa.quantity([-1,2],'arcsec')`.
You can recover the current increments with function increment.

Arguments

Inputs		
value	Increments	
	allowed:	any
	Default:	variant
type	Coordinate type: "direction", "stokes", "spectral", "linear", "tabular". Leave empty for all	
	allowed:	string
	Default:	

Returns
bool

Example

```
"""
#
print "\t----\t setincrement Ex 1 \t----"
csys=cs.newcoordsys(direction=T, spectral=T)
rv = csys.increment(format='q')
print rv
# {'ar_type': 'absolute', 'pw_type': 'world',
# 'quantity': {'*1': {'value': -1.0, 'unit': "'"},
#              '*2': {'value': 1.0, 'unit': "'"},
#              '*3': {'value': 1000.0, 'unit': 'Hz'}}}
rv2 = qa.quantity('4kHz');
csys.setincrement(value=rv2, type='spec')
print csys.increment(type='spec', format='q')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'quantity': {'*1': {'value': 4000.0, 'unit': 'Hz'}}}
csys.setincrement(value='5kHz', type='spec')
print csys.increment(type='spec', format='q')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'quantity': {'*1': {'value': 5000.0, 'unit': 'Hz'}}}
print csys.increment(format='q')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'quantity': {'*1': {'value': -1.0, 'unit': "'"},
#              '*2': {'value': 1.0, 'unit': "'"},
#              '*3': {'value': 5000.0, 'unit': 'Hz'}}}
csys.setincrement (value="-2' 2' 2e4Hz")
print csys.increment(format='q')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'quantity': {'*1': {'value': -2.0, 'unit': "'"},
#              '*2': {'value': 2.0, 'unit': "'"},
#              '*3': {'value': 20000.0, 'unit': 'Hz'}}}
#
"""
```

In the example we first recover the increments as a vector of quantities. We then create a quantity for a new value for the spectral coordinate increment. Note we use units of kHz whereas the spectral coordinate is currently expressed in units of Hz. We then set the increment for the spectral coordinate. We then recover the increment again; you can see 4kHz has been converted to 4000Hz. We also show how to set the increment using a string interface.

coordsys.setlineartransform.html

coordsys.setlineartransform - Function

1.1.3 Set the linear transform

Description

This function set the linear transform component. For Stokes Coordinates this function will return T but do nothing.
You can recover the current linear transform with function lineartransform.

Arguments

Inputs		
type	Coordinate type: ”direction”, ”stokes”, ”spectral”, ”linear”, ”tabular”. Leave empty for all.	
	allowed:	string
	Default:	
value	Linear transform	
	allowed:	any
	Default:	variant

Returns
bool

Example

```
"""
#
print "\t----\t setlineartransform Ex 1 \t----"
csys = cs.newcoordsys(spectral=T, linear=3)
xf = csys.lineartransform('lin')
print xf
#[(1.0, 0.0, 0.0), (0.0, 1.0, 0.0), (0.0, 0.0, 1.0)]
xf[0]=list(xf[0])
```

```
xf[0][1]=0.01
#xf[0]=tuple(xf[0])
csys.setlineartransform('lin',xf)
print csys.lineartransform('lin')
#[(1.0, 0.01, 0.0), (0.0, 1.0, 0.0), (0.0, 0.0, 1.0)]
"""
```

---

coordsys.setnames.html

coordsys.setnames - Function

1.1.3 Set the axis names

Description

Each axis associated with the Coordinate System has a name. It isn't used in any fundamental way.
This function allows you to set new axis names.
You can set the names either for all axes (`type=unset`) or for just the axes associated with a particular coordinate type.
You can recover the current axis names with function names.

Arguments

Inputs		
value	Names	
	allowed:	stringArray
	Default:	
type	Coordinate type: "direction", "stokes", "spectral", "linear", "tabular" or leave empty for all	
	allowed:	string
	Default:	

Returns
bool

Example

```
"""
#
print "\t----\t setnames Ex 1 \t----"
csys = cs.newcoordsys(spectral=T, linear=2)
csys.setnames(value="a b c")
print csys.names()
```

```
#['a', 'b', 'c']
csys.setnames("flying fish", 'lin')
print csys.names()
#['a', 'flying', 'fish']
#
"""
```

---

coordsys.setobserver.html

coordsys.setobserver - Function

1.1.3 Set a new observer

Description

If you want to grab all the glory, or transfer the blame, this function sets a new observer of the observation. You can get the current observer with function observer. The observer's name is not fundamental to the Coordinate System !

Arguments

Inputs		
value	New observer	
	allowed:	string
	Default:	

Returns

bool

Example

```
"""
#
print "\t----\t setobserver Ex 1 \t----"
csys = cs.newcoordsys()
print csys.observer()
#Karl Jansky
csys.setobserver('Ronald Biggs')
print csys.observer()
#Ronald Biggs
#
"""
```

coordsys.setprojection.html

coordsys.setprojection - Function

1.1.3 Set the direction coordinate projection

Description

If the Coordinate System contains a direction coordinate, this function can be used to set the projection. For discussion about celestial coordinate systems, including projections, see the papers by Mark Calabretta and Eric Greisen. The initial draft from 1996 (implemented in CASA) can be found here.
You can use the function projection to find out all the possible types of projection. You can also use it to find out how many parameters you need to describe a particular projection. See Calabretta and Greisen for details about those parameters (see section 4 of their paper); in FITS terms these parameters are what are labelled as PROJP.
Some brief help here on the more common projections in astronomy.

- SIN has either 0 parameters or 2. For coplanar arrays like East-West arrays, one can use what is widely termed the NCP projection. This is actually a SIN projection where the parameters are 0 and $1/tan(\delta_0)$ where δ_0 is the reference declination. Images made from the ATNF's Compact Array with CASA will have such a projection. Otherwise, the SIN projection requires no parameters (but you can give it two each of which is zero if you wish).
- TAN is used widely in optical astronomy. It requires 0 parameters.
- ZEA (zenithal equal area) is used widely in survey work. It requires 0 parameters.

If the Coordinate System does not contain a direction coordinate, an exception is generated.

Arguments

Inputs		
type	Type of projection	
	allowed:	string
	Default:	
parameters	Projection parameters	
	allowed:	doubleArray
	Default:	-1

Returns
bool

Example

```
"""
#
print "\t----\t Ex setprojection 1 \t----"
im = ia.maketestimage('cena',overwrite=true)
csys = ia.coordsys()
print csys.projection()
#{'type': 'SIN', 'parameters': array([ 0.,  0.])}
print csys.projection('ZEA')
#{'nparameters': 0}
csys.setprojection('ZEA')
im2 = ia.regrid('cena.zea', csys=csys.torecord(), overwrite=true)
#
"""

We change the projection of an image from SIN to
ZEA (which requires no parameters).
```

---

coordsys.setreferencecode.html

coordsys.setreferencecode - Function

1.1.3 Set new reference code

Description

This function sets the reference code for the specified coordinate type.
Examples of reference codes are B1950 and J2000 for direction coordinates, or LSRK and BARY for spectral coordinates.
You must specify `type`, selecting from 'direction', or 'spectral' (the first two letters will do). If the Coordinate System does not contain a coordinate of the type you specify, an exception is generated.
Specify the new code with argument `value`. To see the list of possible codes, use the function referencecode (see example).
If `adjust` is T, then the reference value is recomputed. This is invariably the correct thing to do. If `adjust` is F, then the reference code is simply overwritten; do this very carefully.

Arguments

Inputs		
value	Reference code	
	allowed:	string
	Default:	
type	Coordinate type: direction or spectral	
	allowed:	string
	Default:	direction
adjust	Adjust reference value ?	
	allowed:	bool
	Default:	true

Returns
bool

Example

```
"""
#
print "\t----\t Ex setreferencecode 1 \t----"
csys = cs.newcoordsys(direction=T)
clist = csys.referencecode('dir', T)      # See possibilities
print clist
#['J2000', 'JMEAN', 'JTRUE', 'APP', 'B1950', 'BMEAN', 'BTRUE', 'GALACTIC',
# 'HADEC', 'AZEL', 'AZELSW', 'AZELNE', 'AZELGEO', 'AZELSWGEO', 'AZELNEGEO',
# 'JNAT', 'ECLIPTIC', 'MECLIPTIC', 'TECLIPTIC', 'SUPERGAL', 'ITRF', 'TOPO',
# 'ICRS', 'MERCURY', 'VENUS', 'MARS', 'JUPITER', 'SATURN', 'URANUS',
# 'NEPTUNE', 'PLUTO', 'SUN', 'MOON', 'COMET']
print cs.referencecode('dir')
#J2000
cs.setreferencecode('B1950', 'dir', T)
#
"""
```

In this example we first get the list of all possible reference codes for a direction coordinate. Then we set the actual reference code for the direction coordinate in our Coordinate System.

Example

```
"""
#
print "\t----\t Ex setreferencecode 2 \t----"
ia.maketestimage('myimage.j2000',overwrite=true)      # Open image
csys = ia.coordsys()               # Get Coordinate System
print csys.referencecode('dir', F)
#J2000
csys.setreferencecode('B1950', 'dir', T)              # Set new direction system
im2 = ia.regrid(outfile='myimage.b1950', csys=csys.torecord(),
                overwrite=true)  # Regrid and make new image
#
"""
```

```
In this example we show how to regrid an image from J2000
to B1950.  First we recover the Coordinate System  into the Coordsys
\tool\ called {\stf cs}.  We then set a new direction reference code,
making sure we recompute the reference value.  Then the
new Coordinate System is supplied in the regridding process
(done with an Image \tool).
```

---

coordsys.setreferencelocation.html

coordsys.setreferencelocation - Function

1.1.3 Set reference pixel and value

Description

This function sets the reference pixel and reference value to the specified values. The world coordinate can be specified in any of the formats that the output world coordinate is returned in by the toworld function.
You can specify a mask (argument `mask`) indicating which pixel axes are set (T) and which are left unchanged (F). This function will refuse to change the reference location of a Stokes axis (gets you into trouble otherwise).
This function can be rather useful when regridding images. It allows you to keep easily a particular feature centered in the regridded image.

Arguments

Inputs		
pixel	New reference pixel. Defaults to old reference pixel.	
	allowed:	intArray
	Default:	-1
world	New reference value. Defaults to old reference value.	
	allowed:	any
	Default:	variant -1
mask	Indicates which axes to center. Defaults to all.	
	allowed:	boolArray
	Default:	false

Returns
bool

Example

```
"""
#
```

```
print "\t----\t setreferencelocation Ex 1 \t----"
csys = cs.newcoordsys(linear=2)
print csys.referencepixel()
#[0.0, 0.0]
print csys.referencevalue()
#{'numeric': array([ 0.,  0.])}
w = csys.toworld([19,19], format='n')
shp = [128,128]
p = [64, 64]
csys.setreferencelocation (pixel=p, world=w)
print csys.referencepixel()
#[64.0, 64.0]
print csys.referencevalue()
#{'numeric': array([ 19.,  19.])}
#
"""
```

---

coordsys.setreferencepixel.html

coordsys.setreferencepixel - Function

1.1.3 Set the reference pixel

Description

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.
This function allows you to set a new reference pixel. You should not do this on "stokes" axes unless you are an adept or a big risk taker.
You can set the reference pixel either for all axes (`type=unset`) or for just the axes associated with a particular coordinate type.
Bear in mind, that if your Coordinate System came from a real image, then the reference pixel is special and you should not change it for Direction Coordinates.
You can recover the current reference pixel with function referencepixel.

Arguments

Inputs	
value	Reference pixel
	allowed: doubleArray
	Default:
type	Coordinate type: "direction", "stokes", "spectral", "linear", "tabular" or leave unset for all
	allowed: string
	Default:

Returns
bool

Example

"""

```
#
print "\t----\t setreferencepixel Ex 1 \t----"
csys = cs.newcoordsys(spectral=T, linear=2)
csys.setreferencepixel(value=[1.0, 2.0, 3.0])
print csys.referencepixel()
#[1.0, 2.0, 3.0]
csys.setreferencepixel([-1, -1], 'lin')
print csys.referencepixel()
#[1.0, -1.0, -1.0]
#
"""
```

---

coordsys.setreferencevalue.html

coordsys.setreferencevalue - Function

1.1.3 Set the reference value

Description

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.
This function allows you to set a new reference value. You should not do this on "stokes" axes unless you are an adept or a big risk taker.
You may supply the reference value in all of the formats described in the formatting discussion.
You can recover the current reference value with function referencevalue.
Note that the value argument should be one of the specified possibilitioes. Especially a `measure` will be accepted, but will have a null effect, due to the interpretation as a generic record.

Arguments

Inputs		
value	Reference value	
	allowed:	any
	Default:	variant
type	Coordinate type: "direction", "stokes", "spectral", "linear", "tabular" or leave empty for all.	
	allowed:	string
	Default:	

Returns
bool

Example

"""

```
#
print "\t----\t setreferencevalue Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
rv = csys.referencevalue(format='q')
print rv
#{'quantity': {'*1': {'value': 0.0, 'unit': "'"},
# '*2': {'value': 0.0, 'unit': "'"}, '*3': {'value': 1415000000.0, 'unit': 'Hz'}}}
rv2 = rv['quantity']['*3']
rv2['value'] = 2.0e9
print rv2
#{'value': 2000000000.0, 'unit': 'Hz'}
csys.setreferencevalue(type='spec', value=rv2)
print csys.referencevalue(format='n')
#{'numeric': array([  0.00000000e+00,   0.00000000e+00,   2.00000000e+09])}
#
# To set a new direction reference value, the easiest way, given a
# direction measure dr would be:
dr = me.direction('j2000','30deg','40deg')
# SHOULD BE SIMPLIFIED!!!
newrv=csys.referencevalue(format='q')
newrv['quantity']['*1']=dr['m0']
newrv['quantity']['*2']=dr['m1']
csys.setreferencevalue(value=newrv)
print csys.referencevalue(format='q')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'quantity': {'*1': {'value': 1800.0, 'unit': "'"},
#              '*2': {'value': 2399.9999999999995, 'unit': "'"},
#              '*3': {'value': 1415000000.0, 'unit': 'Hz'}}}
#
"""
```

---

coordsys.setrestfrequency.html

coordsys.setrestfrequency - Function

1.1.3 Set the rest frequency

Description

If the Coordinate System contains a spectral coordinate, then it has a rest frequency. In fact, the spectral coordinate can hold several rest frequencies (to handle for example, an observation where the band covers many lines), although only one is active (for velocity conversions) at a time.
This function allows you to set new rest frequencies. You can provide the rest frequency as a quantity, or as a quantity string, or a double (units of current rest frequency assumed).
You specify whether the list of frequencies will be appended to the current list or whether it will replace that list. You must select which of the frequencies will become the active one. By default its the first in the list. The index refers to the final list (either appended or replaced).
You can recover the current rest frequencies with restfrequency.
If the Coordinate System does not contain a frequency coordinate, an exception is generated.

Arguments

Inputs		
value	New rest frequencies	
	allowed:	any
	Default:	variant
which	Which is the active rest frequency	
	allowed:	int
	Default:	0
append	Append this list or overwrite ?	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
"""
#
print "\t----\t setrestfrequency Ex 1 \t----"
csys = cs.newcoordsys(spectral=T)
print csys.restfrequency()
#{'value': array([  1.42040575e+09]), 'unit': 'Hz'}
csys.setrestfrequency(qa.quantity('1.4GHz'))
print csys.restfrequency()
#{'value': array([  1.40000000e+09]), 'unit': 'Hz'}
csys.setrestfrequency(1.3e9)
print csys.restfrequency()
#{'value': array([  1.30000000e+09]), 'unit': 'Hz'}
csys.setrestfrequency (value=[1.2e9, 1.3e9], which=1)
print csys.restfrequency()
#{'value': array([  1.30000000e+09,   1.20000000e+09]), 'unit': 'Hz'}
csys.setrestfrequency (qa.quantity([1,2],'GHz'), which=3, append=T)
print csys.restfrequency()
#{'value': array([  2.00000000e+09,   1.20000000e+09,   1.30000000e+09,
#         1.00000000e+09]), 'unit': 'Hz'}
csys.setrestfrequency ("1.4E9Hz 1667MHz")
print csys.restfrequency()
#{'value': array([  1.40000000e+09,   1.66700000e+09]), 'unit': 'Hz'}
#
"""
```

---

coordsys.setspectral.html

coordsys.setspectral - Function

1.1.3 Set tabular values for the spectral coordinate

Description

When you construct a Coordsys `tool`, if you include a Spectral Coordinate, it will be linear in frequency. This function allows you to replace the Spectral Coordinate by a finite table of values. Coordinate conversions between pixel and world are then done by interpolation.
You may specify either a vector of frequencies or velocities. If you specify frequencies, you can optionally specify a (new) reference code (see function setreferencecode for more details) and rest frequency (else the existing ones will be used).
If you specify velocities, you can optionally specify a (new) reference code and rest frequency (else the existing ones will be used). You must also give the doppler type (see function summary for more details). The velocities are then converted to frequency for creation of the Spectral Coordinate (which is fundamentally described by frequency).
You may specify the rest frequency as a Quantum or a double (native units of Spectral Coordinate used).

Arguments

Inputs	
refcode	Reference code. Leave unset for no change.
	allowed: string
	Default:
restfreq	Rest frequency. Leave unset for no change.
	allowed: any
	Default: variant
frequencies	Vector of frequencies. Leave unset for no change.
	allowed: any
	Default: variant 1GHz
doppler	Doppler type. Leave unset for no change.
	allowed: string
	Default:
velocities	Vector of velocities types. Leave unset for no change.
	allowed: any
	Default: variant 1km/s

Returns
bool

Example

```
print "\t----\t setspectral Ex 1 \t----"
csys = cs.newcoordsys(spectral=T);
f1 = [1,1.01,1.03,1.4]
fq = qa.quantity(f1, 'GHz')
csys.setspectral(frequencies=fq)
v = csys.frequencytovelocity(f1, 'GHz', 'radio', 'km/s')
print 'v=', v
#v= [88731.317461076716, 86620.706055687479, 82399.483244909003, 4306.8612455073862]
vq = qa.quantity(v, 'km/s')
csys.setspectral(velocities=vq, doppler='radio')
f2 = csys.velocitytofrequency(v, 'GHz', 'radio', 'km/s')
print 'f1 = ', f1
#f1 =  [1, 1.01, 1.03, 1.3999999999999999]
print 'f2 = ', f2
#f2 =  [1.0, 1.01, 1.03, 1.3999999999999999]

We make a linear Spectral Coordinate.  Then overwrite it with
a list of frequenices.  Convert those values to velocity,
then overwrite the coordinate starting with a list of
velocities. Then convert the velocities to frequency
and show we get the original result.
```

---

coordsys.setstokes.html

coordsys.setstokes - Function

1.1.3 Set the Stokes types

Description

If the Coordinate System contains a Stokes Coordinate, this function allows you to change the Stokes types defining it. If there is no Stokes Coordinate, an exception is generated.
See the coordsys constructor to see the possible Stokes types you can set.
You can set the Stokes types with function setstokes.

Arguments

Inputs		
stokes	Stokes types	
	allowed:	stringArray
	Default:	

Returns
bool

Example

```
"""
#
print "\t----\t setstokes Ex 1 \t----"
csys = cs.newcoordsys(stokes="I V")
print csys.stokes()
#['I', 'V']
csys.setstokes("XX RL")
print csys.stokes()
#['XX', 'RL']
#
"""
```

coordsys.settabular.html

coordsys.settabular - Function

1.1.3 Set tabular values for the tabular coordinate

Description

When you construct a Coordsys `tool`, if you include a Tabular Coordinate, it will be linear. This function allows you to replace the Tabular Coordinate by a finite table of values. Coordinate conversions between pixel and world are then done by interpolation (or extrapolation beyond the end). The table of values must be at least of length 2 or an exception will occur.
You may specify a vector of pixel and world values (in the current units of the Tabular Coordinate). These vectors must be the same length. If you leave one of them unset, then the old values are used, but again, ultimately, the pixel and world vectors must be the same length.
The new reference pixel will be the first pixel value. The new reference value will be the first world value.
Presently, there is no way for you to recover the lookup table once you have set it.
If you have more than one Tabular Coordinate, use argument `which` to specify which one you want to modify.

Arguments

Inputs		
pixel	Vector of (0-rel) pixel values. Default is no change.	
	allowed:	doubleArray
	Default:	-1
world	Vector of world values. Default is no change.	
	allowed:	doubleArray
	Default:	-1
which	Which Tabular coordinate	
	allowed:	int
	Default:	0

Returns
bool

Example

```
"""
#
print "\t----\t settabular Ex 1 \t----"
csys = cs.newcoordsys(tabular=T);
print csys.settabular (pixel=[1,10,15,20,100], world=[10,20,50,100,500])
#True
#
"""

We make a linear Tabular Coordinate.  Then overwrite it with
a non-linear list of pixel and world values.
```

---

coordsys.settelescope.html

coordsys.settelescope - Function

1.1.3 Set a new telescope

Description

This function sets a new telescope of the observation. The telescope position may be needed for reference code conversions; this is why it is maintained in the Coordinate System. So it is fundamental to the Coordinate System and should be correct.
You can find a list of the observatory names know to CASA with the Measures obslist function.
You can get the current telescope with function telescope.

Arguments

Inputs		
value	New telescope	
	allowed:	string
	Default:	

Returns
bool

Example

```
"""
#
print "\t----\t settelescope Ex 1 \t----"
csys = cs.newcoordsys()
print csys.telescope()
#ATCA
csys.settelescope('VLA')
print csys.telescope()
#VLA
```

```
csys.settelescope('The One In My Backyard')
#Tue Mar 6 21:41:24 2007      WARN coordsys::settelescope:
#This telescope is not known to the casapy system
#You can request that it be added
print me.obslist()
#ALMA ARECIBO ATCA BIMA CLRO DRAO DWL GB GBT GMRT IRAM PDB IRAM_PDB
# JCMT MOPRA MOST NRAO12M NRAO_GBT PKS SAO SMA VLA VLBA WSRT
#
"""
```

---

coordsys.setunits.html

coordsys.setunits - Function

1.1.3 Set the axis units

Description

Each axis associated with the Coordinate System has a unit. This function allows you to set new axis units.
You can set the units either for all axes (`type=unset`) or for just the axes associated with a particular coordinate type.
In general, the units must be consistent with the old units. When you change the units, the increment and reference value will be adjusted appropriately. However, for a linear or tabular coordinate, and only when you specify `type='linear'` or `type='tabular'` (i.e. you supply units only for the specified linear of tabular coordinate), and if you set `overwrite=T`, you can just overwrite the units with no further adjustments. Otherwise, the `overwrite` argument will be silently ignored. Use argument `which` to specify which coordinate if you have more than one of the specified type.
You can recover the current axis units with function units.

Arguments

Inputs		
value	Units	
	allowed:	stringArray
	Default:	
type	Coordinate type: "direction", "stokes", "spectral", "linear", "tabules" or leave unset for all.	
	allowed:	string
	Default:	
overwrite	Overwrite linear or tabular coordinate units?	
	allowed:	bool
	Default:	false
which	Which coordinate if more than one of same type. Default is first.	
	allowed:	int
	Default:	-10

Returns

bool

Example

```
"""
#
print "\t----\t setunits Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
csys.summary()
csys.setunits(value="deg rad mHz");
csys.summary()
#
"""
```

---

coordsys.stokes.html

coordsys.stokes - Function

1.1.3 Recover the Stokes types

Description

If the Coordinate System contains a Stokes Coordinate, this function recovers the Stokes types defining it. If there is no Stokes Coordinate, an exception is generated.
You can set the Stokes types with function setstokes.

Arguments

Returns
stringArray

Example

```
"""
#
print "\t----\t stokes Ex 1 \t----"
csys = cs.newcoordsys(stokes=['I','V'])
print csys.stokes()
#['I', 'V']
csys = cs.newcoordsys(stokes='Q U')
print csys.stokes()
#['Q', 'U']
#
"""
```

coordsys.summary.html

coordsys.summary - Function

1.1.3 Summarize basic information about the Coordinate System

Description

This function summarizes the information contained in the Coordinate System. For spectral coordinates, the information is listed as a velocity as well as a frequency. The argument `doppler` allows you to specify what doppler convention it is listed in. You can choose from `radio, optical` and `beta`. Alternative names are `z` for `optical`, and `relativistic` for `beta`. The default is `radio`. The definitions are

- radio: $1 - F$
- optical: $-1 + 1/F$
- beta: $(1 - F^2)/(1 + F^2)$

where $F = \nu/\nu_0$ and ν_0 is the rest frequency. If the rest frequency has not been set in your image, you can set it with the function setrestfrequency. These velocity definitions are provided by the measures system via the Doppler measure (see example).
If you set `list=F`, then the summary will not be written to the global logger. However, the return value will be a vector of strings holding the summary information, one string per line of the summary.
For direction and spectral coordinates, the reference frame (e.g. J2000 or LSRK) is also listed. Along side this, in parentheses, will be the conversion reference frame as well (if it is different from the native reference frame). See function setconversion to see what this means.

Arguments

Inputs		
doppler	List velocity information with this doppler definition	
	allowed:	string
	Default:	RADIO
list	List to global logger	
	allowed:	bool
	Default:	true

Returns
stringArray

Example

```
"""
#
print "\t----\t summary Ex 1 \t----"
d = me.doppler('beta')
print me.listcodes(d)
#[normal=RADIO Z RATIO BETA GAMMA OPTICAL TRUE RELATIVISTIC, extra=]
csys = cs.newcoordsys(direction=T, spectral=T)
print csys.summary(list=F)
#
#Direction reference : J2000
#Spectral  reference : LSRK
#Velocity  type      : RADIO
#Rest frequency      : 1.42041e+09 Hz
#Telescope           : ATCA
#Observer            : Karl Jansky
#Date observation    : 2007/07/14/04:49:31
#
#Axis Coord Type      Name             Proj   Coord value at pixel    Coord incr Units
#-------------------------------------------------------------------------------------
#0    0     Direction Right Ascension   SIN  00:00:00.000     0.00 -6.000000e+01 arcsec
#1    0     Direction Declination       SIN +00.00.00.000     0.00  6.000000e+01 arcsec
#2    1     Spectral  Frequency                 1.415e+09     0.00  1.000000e+03 Hz
#                     Velocity                    1140.94     0.00 -2.110611e-01 km/s
#
#
"""
```

coordsys.telescope.html

coordsys.telescope - Function

1.1.3 Return the telescope

Description

This function returns the telescope contained in the Coordinate System as a simple string.
The telescope position may be needed for reference code conversions; this is why it is maintained in the Coordinate System.
The conversion from string to position is done with Measures observatory. The example shows how.

Arguments

Returns
string

Example

```
"""
#
print "\t----\t telescope Ex 1 \t----"
csys = cs.newcoordsys()
print csys.telescope()
#ATCA
print me.observatory(csys.telescope())
#{'type': 'position', 'refer': 'ITRF',
# 'm1': {'value': -0.5261379196128062, 'unit': 'rad'},
# 'm0': {'value': 2.6101423190348916, 'unit': 'rad'},
# 'm2': {'value': 6372960.2577234386, 'unit': 'm'}}
#
```

```
"""

We get the telescope as a string.
 The Measures system is used to convert from
the simple name to a position Measure.
```

---

coordsys.toabs.html

coordsys.toabs - Function

1.1.3 Convert relative coordinate to absolute

Description

This function converts a relative coordinate to an absolute coordinate. The coordinate may be a pixel coordinate or a world coordinate.
If the coordinate is a pixel coordinate, it is supplied as a numeric vector. If the coordinate is a world coordinate, you may give it in all of the formats described in the formatting discussion.
If the coordinate value is supplied by a Coordsys `tool` function (e.g. toworld) then the coordinate 'knows' whether it is world or pixel (and absolute or relative). However, you might supply the value from some other source as a numeric vector (which could be world or pixel) in which case you must specify whether it is a world or pixel coordinate via the `isworld` argument.

Arguments

Inputs		
value	Relative coordinate	
	allowed:	any
	Default:	variant
isworld	Is coordinate world or pixel? Default is unset.	
	allowed:	int
	Default:	-1

Returns
record

Example

```
"""
#
print "\t----\t toabs Ex 1 \t----"
```

```
csys = cs.newcoordsys(direction=T, spectral=T)
aw = csys.toworld([100,100,24], 's')
rw = csys.torel(aw)
aw2 = csys.toabs(rw)
print aw
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'string': array(['23:53:19.77415678', '+01.40.00.84648186',
#                  '1.41502400e+09 Hz'], dtype='|S19')}
print rw
#{'ar_type': 'relative', 'pw_type': 'world',
# 'string': array(['-6.00084720e+03 arcsec', '6.00084648e+03 arcsec',
#                  '2.40000000e+04 Hz'], dtype='|S23')}
print aw2
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'string': array(['23:53:19.77415672', '+01.40.00.84648000',
#                  '1.41502400e+09 Hz'], dtype='|S19')}
#
"""
```

This example uses world coordinates.

Example

```
"""
#
print "\t----\t toabs Ex 2 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
ap = csys.topixel()           # Reference value
rp = csys.torel(ap)
ap2 = csys.toabs(rp)
print ap
#{'ar_type': 'absolute', 'pw_type': 'pixel', 'numeric': array([ 0.,  0.,  0.])}
print rp
#{'ar_type': 'relative', 'pw_type': 'world',
#     'numeric': array([  0.00000000e+00,   0.00000000e+00,  -1.41500000e+09])}
print ap2
#{'ar_type': 'absolute', 'pw_type': 'world', 'numeric': array([ 0.,  0.,  0.])}
#
"""
```

```
This example uses pixel coordinates.
```

---

coordsys.toabsmany.html

coordsys.toabsmany - Function

1.1.3 Convert many numeric relative coordinates to absolute

Description

This function converts many relative coordinates to absolute. It exists so you can efficiently make many conversions (which would be rather slow if you did them all with toabs). Because speed is the object, the interface is purely in terms of numeric matrices, rather than being able to accept strings and quanta etc. like toabs can.
When dealing with world coordinates, the units of the numeric values must be the native units, given by function units.

Arguments

Inputs		
value	Relative coordinates	
	allowed:	any
	Default:	variant
isworld	Is coordinate world or pixel? Default is unset.	
	allowed:	int
	Default:	-1

Returns
record

Example

```
"""
#
print "\t----\t toabsmany Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)    # 3 axes
rv = csys.referencevalue();                 # reference value
w = csys.torel(rv)                          # make relative
```

```
inc = csys.increment();                      # increment
off=[]
for idx in range(100):
  off.append(inc['numeric'][2]*idx)          # offset for third axis
wrel = ia.makearray(0,[3,100])               # 100 conversions each of length 3
for i in range(3):
  for j in range(100):
    wrel[i][j]=w['numeric'][i]
for j in range(100):
  wrel[2][j] += off[j]                       # Make spectral axis values change
wabs  = csys.toabsmany (wrel, T)['numeric'] # Convert
print wabs[0][0],wabs[1][0],wabs[2,0]        # First absolute coordinate
#0.0 0.0 1415000000.0
print wabs[0][99],wabs[1][99],wabs[2][99]   # 100th absolute coordinate
#0.0 0.0 1415099000.0
#
"""

This example uses world coordinates.
```

---

coordsys.topixel.html

coordsys.topixel - Function

1.1.3 Convert from absolute world to pixel coordinate

Description

This function converts between world (physical) coordinate and absolute pixel coordinate (0-rel).
The world coordinate can be provided in one of four formats via the argument `world`. These match the output formats of function toworld.
If you supply fewer world values than there are axes in the Coordinate System, your coordinate vector will be padded out with the reference value for the missing axes. Excess values will be silently ignored.
You may supply the world coordinate in all of the formats described in the formatting discussion.

Arguments

Inputs		
value	Absolute world coordinate	
	allowed:	any
	Default:	variant

Returns
record

Example

```
"""
#
print "\t----\t topixel Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T, stokes="I V", linear=2)
w = csys.toworld([-2,2,1,2,23,24], 'n')
print csys.topixel(w)
#{'ar_type': 'absolute', 'pw_type': 'pixel',
```

```
# 'numeric': array([ -2.,   2.,   1.,   2.,  23.,  24.])}
w = csys.toworld([-2,2,1,2,23,24], 'q')
print csys.topixel(w)
#{'ar_type': 'absolute', 'pw_type': 'pixel',
# 'numeric': array([ -2.,   2.,   1.,   2.,  23.,  24.])}
w = csys.toworld([-2,2,1,2,23,24], 'm')
print csys.topixel(w)
#{'ar_type': 'absolute', 'pw_type': 'pixel',
# 'numeric': array([ -2.,   2.,   1.,   2.,  23.,  24.])}
w = csys.toworld([-2,2,1,2,23,24], 's')
print cs.topixel(w)
#{'ar_type': 'absolute', 'pw_type': 'pixel',
# 'numeric': array([ -2.,   2.,   1.,   2.,  23.,  24.])}
w = csys.toworld([-2,2,1,2,23,24], 'mnq')
print cs.topixel(w)
#{'ar_type': 'absolute', 'pw_type': 'pixel',
# 'numeric': array([ -2.,   2.,   1.,   2.,  23.,  24.])}
#
"""
```

Example

```
"""
#
print "\t----\t topixel Ex 2 \t----"
csys = cs.newcoordsys (stokes="I V", linear=2)
print csys.toworld([0,1,2], 's')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'string': array(['I', '1.00000000e+00 km', '2.00000000e+00 km'],
#  dtype='|S18')}
print csys.toworld([0,1,2], 'm')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'measure': {'stokes': 'I', 'linear': {'*1': {'value': 1.0, 'unit': 'km'},
# '*2': {'value': 2.0, 'unit': 'km'}}}}
print csys.toworld([0,1,2], 'q')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'quantity': {'*1': {'value': 1.0, 'unit': ''},
# '*2': {'value': 1.0, 'unit': 'km'}, '*3': {'value': 2.0, 'unit': 'km'}}}
#
"""
```

Example

```
"""
#
print "\t----\t topixel Ex 3 \t----"
csys = cs.newcoordsys (spectral=T, linear=1)
print csys.toworld([0,1,2], 'q')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'quantity': {'*1': {'value': 1415000000.0, 'unit': 'Hz'},
# '*2': {'value': 1.0, 'unit': 'km'}}}
#
"""
```

---

coordsys.topixelmany.html

coordsys.topixelmany - Function

1.1.3 Convert many absolute numeric world coordinates to pixel

Description

This function converts many absolute world coordinates to pixel coordinates. It exists so you can efficiently make many conversions (which would be rather slow if you did them all with topixel). Because speed is the object, the interface is purely in terms of numeric matrices, rather than being able to accept strings and quanta etc. like topixel can.
The units of the numeric values must be the native units, given by function units.

Arguments

Inputs		
value	Absolute world coordinates	
	allowed:	any
	Default:	variant

Returns
record

Example

```
"""
#
print "\t----\t topixelmany Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)    # 3 axes
rv = csys.referencevalue();                 # reference value
inc = csys.increment();                     # increment
off = []
for idx in range(100):
  off.append(inc['numeric'][2] * idx)       # offset for third axis
```

```
wabs = ia.makearray(0, [3,100])             # 100 conversions each of length 3
for i in range(3):
  for j in range(100):
    wabs[i][j]=rv['numeric'][i]
for j in range(100):
  wabs[2][j] += off[j]                      # Make spectral axis values change
pabs  = csys.topixelmany (wabs)['numeric']; # Convert
print pabs[0][0], pabs[1][0], pabs[1][2]    # First absolute pixel coordinate
#0.0 0.0 0.0
print pabs[0][99], pabs[1][99], pabs[2][99] # 100th absolute pixel coordinate
#0.0 0.0 99.0
#
"""
```

---

coordsys.torecord.html

coordsys.torecord - Function

1.1.3 Convert Coordinate System to a record

Description

You can convert a Coordinate System to a record with this function. There is also fromrecord to set a Coordinate System from a record.
These functions allow Coordsys `tools` to be used as parameters in the methods of other tools.

Arguments

Returns
record

Example

```
"""
#
print "\t----\t torecord Ex 1 \t----"
csys = cs.newcoordsys(direction=T, stokes="I Q")
rec = csys.torecord();
cs2 = cs.newcoordsys();
print cs2.ncoordinates()
#0
cs2.fromrecord(rec);
print csys.ncoordinates(), cs2.ncoordinates()
#2 2
#
"""
```

coordsys.subimage.html

coordsys.subimage - Function

1.1.3 delivers a coordinate origin re-referenced for a subimage

Description

You can convert a Coordinate System to another coordinatesystem applicable to a subImage. The newshape does not matter as this is the coordinatesystem not the image except for Stokes axis; therefore you can ignore `newshape` except when your sub-image you are considering has only a section of your original Stokes axis.

Arguments

Inputs		
originshft	The shift value from original reference (vector of values in pixels)	
	allowed:	any
	Default:	variant
newshape	The new shape of the image it will applicable to (pixel shape)	
	allowed:	intArray
	Default:	

Returns

record

Example

```
"""
#
print "\t----\t subimage Ex 1 \t----"
ia.open('original.image')
csys = ia.coordsys()
imshape=ia.shape()
```

```
#want to make an empty sub image of the 11th channel
#keeping other reference pixel as is
refshft=[0,0,0,10]
subcoordsysrec=csys.subimage(neworigin=refshft)
imshape[3]=1
ia.fromshape(outfile='Eleventh_chan_template.image', shape=imshape, csys=subcoordsysrec)

"""
```

---

coordsys.torel.html

coordsys.torel - Function

1.1.3 Convert absolute coordinate to relative

Description

This function converts an absolute coordinate to a relative coordinate. The coordinate may be a pixel coordinate or a world coordinate.
Relative coordinates are relative to the reference pixel (pixel coordinates) or the reference value (world coordinates) in the sense
$relative = absolute - reference$.
If the coordinate is a pixel coordinate, it is supplied as a numeric vector. If the coordinate is a world coordinate, you may give it in all of the formats described in the formatting discussion.
If the coordinate value is supplied by a Coordsys `tool` function (e.g. toworld) then the coordinate 'knows' whether it is world or pixel (and absolute or relative). However, you might supply the value from some other source as a numeric vector (which could be world or pixel) in which case you must specify whether it is a world or pixel coordinate via the `isworld` argument.

Arguments

Inputs		
value	Absolute coordinate	
	allowed:	any
	Default:	variant
isworld	Is coordinate world or pixel? Default is unset.	
	allowed:	int
	Default:	-1

Returns
record

Example

```
"""
#
print "\t----\t torel Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
aw = csys.toworld([99,99,23], 's')
rw = csys.torel(aw)
aw2 = csys.toabs(rw)
print aw
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'string': array(['23:53:23.78086843', '+01.39.00.82133427',
# '1.41502300e+09 Hz'], dtype='|S19')}
print rw
#{'ar_type': 'relative', 'pw_type': 'world',
# 'string': array(['-5.94082202e+03 arcsec', '5.94082133e+03 arcsec',
# '2.30000000e+04 Hz'], dtype='|S23')}
print aw2
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'string': array(['23:53:23.78086818', '+01.39.00.82133000',
# '1.41502300e+09 Hz'], dtype='|S19')}
#
"""
```

This example uses world coordinates.

```
"""
#
print "\t----\t torel Ex 2 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
ap = csys.topixel()           # Reference value
rp = csys.torel(ap)
ap2 = csys.toabs(rp)
print ap
#{'ar_type': 'absolute', 'pw_type': 'pixel', 'numeric': array([ 0.,  0.,  0.])}
print rp
#{'ar_type': 'relative', 'pw_type': 'pixel', 'numeric': array([ 0.,  0.,  0.])}
print ap2
#{'ar_type': 'absolute', 'pw_type': 'pixel', 'numeric': array([ 0.,  0.,  0.])}
#
"""
```

This example uses pixel coordinates.

coordsys.torelmany.html

coordsys.torelmany - Function

1.1.3 Convert many numeric absolute coordinates to relative

Description

This function converts many absolute coordinates to relative. It exists so you can efficiently make many conversions (which would be rather slow if you did them all with torel). Because speed is the object, the interface is purely in terms of numeric matrices, rather than being able to accept strings and quanta etc. like torel can.
When dealing with world coordinates, the units of the numeric values must be the native units, given by function units.

Arguments

Inputs		
value	Absolute coordinates	
	allowed:	any
	Default:	variant
isworld	Is coordinate world or pixel? Default is unset.	
	allowed:	int
	Default:	-1

Returns
record

Example

```
"""
#
print "\t----\t torelmany Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)    # 3 axes
w = csys.referencevalue();                  # reference value
inc = csys.increment();                     # increment
```

```
off = []
for idx in range(100):
  off.append(inc['numeric'][2] * idx)       # offset for third axis
wabs = ia.makearray(0, [3,100])             # 100 conversions each of length 3
for i in range(3):
  for j in range(100):
    wabs[i][j] = w['numeric'][i]
for j in range(100):
  wabs[2][j] += off[j]                      # Make spectral axis values change
wrel  = cs.torelmany (wabs, T)['numeric']   # Convert
print wrel[0][0], wrel[1][0], wrel[2][0]    # First relative coordinate
#0.0 0.0 0.0
print wrel[0][99], wrel[1][99], wrel[2][99] # 100th relative coordinate
#0.0 0.0 99000.0
#
"""
```

This example uses world coordinates.

---

coordsys.toworld.html

coordsys.toworld - Function

1.1.3 Convert from absolute pixel coordinate to world

Description

This function converts between absolute pixel coordinate (0-rel) and absolute world (physical coordinate).
If you supply fewer pixel values than there are axes in the Coordinate System, your coordinate vector will be padded out with the reference pixel for the missing axes. Excess values will be silently ignored.
You may ask for the world coordinate in all of the formats described in the discussion regarding the formatting possibilities available via argument `format`.

Arguments

Inputs		
value	Absolute pixel coordinate. Default is reference pixel.	
	allowed:	any
	Default:	variant
format	Format string:	combination of ”n”, ”q”, ”s”, ”m”
	allowed:	string
	Default:	n

Returns
record

Example

```
"""
#
print "\t----\t toworld Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
```

```
print csys.toworld([-3,1,1], 'n')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'numeric': array([  3.00000051e+00,   1.00000001e+00,   1.41500100e+09])}
print csys.toworld([-3,1,1], 'q')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'quantity': {'*1': {'value': 3.0000005076962117, 'unit': "'"},
#              '*2': {'value': 1.0000000141027674, 'unit': "'"},
#              '*3': {'value': 1415001000.0, 'unit': 'Hz'}}}
print csys.toworld([-3,1,1], 'm')
#{'ar_type': 'absolute', 'pw_type': 'world', 'measure':
# {'spectral': {'radiovelocity': {'type': 'doppler', 'm0': {'value': 1140733.0762829871, 'un
#             'opticalvelocity': {'type': 'doppler', 'm0': {'value': 1145090.2316004676, 'un
#                 'frequency': {'type': 'frequency', 'm0': {'value': 1415001000.0, 'unit':
#               'betavelocity': {'type': 'doppler', 'm0': {'value': 1142903.3485169839, 'un
# 'direction': {'type': 'direction', 'm1': {'value': 0.0002908882127680503, 'unit': 'rad'},
#                                    'm0': {'value': 0.00087266477368000634, 'unit': 'rad'}
print csys.toworld([-3,1,1], 's')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'string': array(['00:00:12.00000203', '+00.01.00.00000085', '1.41500100e+09 Hz'], dtype='|
#
"""
```

Example

```
"""
#
print "\t----\t toworld Ex 2 \t----"
csys = cs.newcoordsys (stokes="I V", linear=2)
print csys.toworld([0,1,2], 's')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'string': array(['I', '1.00000000e+00 km', '2.00000000e+00 km'],
#      dtype='|S18')}
print csys.toworld([0,1,2], 'm')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'measure': {'stokes': 'I', 'linear': {'*1': {'value': 1.0, 'unit': 'km'},
#                                       '*2': {'value': 2.0, 'unit': 'km'}}}}
print csys.toworld([0,1,2], 'q')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'quantity': {'*1': {'value': 1.0, 'unit': ''},
#              '*2': {'value': 1.0, 'unit': 'km'},
```

```
#              '*3': {'value': 2.0, 'unit': 'km'}}}
#
"""
```

Example

```
"""
#
print "\t----\t toworld Ex 3 \t----"
csys = cs.newcoordsys (spectral=T, linear=1)
print cs.toworld([0,1,2], 'q')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'quantity': {'*1': {'value': 1415000000.0, 'unit': 'Hz'},
#              '*2': {'value': 1.0, 'unit': 'km'}}}
#
"""
```

---

coordsys.toworldmany.html

coordsys.toworldmany - Function

1.1.3 Convert many absolute pixel coordinates to numeric world

Description

This function converts many absolute pixel coordinates to world coordinates. It exists so you can efficiently make many conversions (which would be rather slow if you did them all with toworld). Because speed is the object, the interface is purely in terms of numeric matrices, rather than being able to produce strings and quanta etc. like toworld can.
The units of the output world values are the native units given by function units.

Arguments

Inputs		
value	Absolute pixel coordinates	
	allowed:	variant
	Default:	

Returns

record

Example

```
"""
#
print "\t----\t toworldmany Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)    # 3 axes
rp = csys.referencepixel()['numeric'];            # reference pixel
pabs = ia.makearray(0,[3,100])              # 100 conversions each of length 3
for i in range(3):
  for j in range(100):
    pabs[i][j] = rp[i]
```

```
for ioff in range(100):                      # offset for third axis
  pabs[2][ioff] += ioff;                     # Make spectral axis values change
wabs  = csys.toworldmany (pabs)['numeric']; # Convert
print wabs[0][0], wabs[1][0], wabs[2][0]    # First absolute pixel coordinate
#0.0 0.0 1415000000.0
print wabs[0][99], wabs[1][99], wabs[2][99] # 100th absolute pixel coordinate
#0.0 0.0 1415099000.0
#
"""
```

---

coordsys.type.html

coordsys.type - Function

1.1.3 Return the type of this tool

Description

This function returns the string ‘coordsys’.

Arguments

Returns
string

---

coordsys.units.html

coordsys.units - Function

1.1.3 Recover the units for each axis

Description

Each axis associated with the Coordinate System has a unit. This function returns those units (in world axis order).
You can recover the units either for all coordinates (leave `type` unset) or for a specific coordinate type (mimumum match of the allowed types will do). If you ask for a non-existent coordinate an exception is generated.
You can set the units with function setunits.

Arguments

Inputs	
type	Coordinate type: 'direction', 'stokes', 'spectral', 'linear' or leave unset for all allowed: string Default:

Returns
stringArray

Example

```
"""
#
print "\t----\t units Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
print csys.units()
#["'", "'", 'Hz']
print csys.units('spec')
#Hz
#
```

"""

---

coordsys.velocitytofrequency.html

coordsys.velocitytofrequency - Function

1.1.3 Convert velocity to frequency

Description

This function converts velocities to frequencies.
The input velocities are specified via a vector of numeric values, a specified unit (`velunit`), and a velocity doppler definition (`doppler`).
The frequencies are returned in a vector for which you specify the units (`frequnit`). If you don't give the unit, it is assumed that the units are those given by function units for the spectral coordinate.
This function will return a fail if there is no spectral coordinate in the Coordinate System. See also the function frequencytovelocity.

Arguments

Inputs		
value	Velocity to convert	
	allowed:	doubleArray
	Default:	
frequnit	Unit of output frequencies. Default is intrinisic units.	
	allowed:	string
	Default:	
doppler	Velocity doppler definition	
	allowed:	string
	Default:	radio
velunit	Unit of input velocities	
	allowed:	string
	Default:	km/s

Returns
doubleArray

Example

```
"""
#
print "\t----\t velocitytofrequency Ex 1 \t----"
ia.fromshape('hcn.cube',[64,64,32,4], overwrite=true)
csys = ia.coordsys()
rtn = csys.findcoordinate('spectral')   # Find spectral axis
pixel = csys.referencepixel();          # Use reference pixel for non-spectral
pa = rtn['pixel']
wa = rtn['world']
nFreq = ia.shape()[pa]                  # Length of spectral axis
freq = []
for i in range(nFreq):
  pixel[pa] = i;                        # Assign value for spectral axis of pixel coordinate
  w = csys.toworld(value=pixel, format='n')# Convert pixel to world
  freq.append(w['numeric'][wa])         # Fish out frequency
print "freq=", freq
vel = csys.frequencytovelocity(value=freq, doppler='optical', velunit='km/s')
freq2 = csys.velocitytofrequency(value=vel, doppler='optical', velunit='km/s')
print "vel=",vel
print "freq2=",freq2
csys.done()
#
exit() # This is last example so exit casapy if you wish.
#
"""
```

In this example, we find the optical velocity in km/s of every pixel along the spectral axis of our image. First we obtain the Coordinate System from the image. Then we find which axis of the Coordinate System (image) pertain to the spectral coordinate. Then we loop over each pixel of the spectral axis, and convert a pixel coordinate (one for each axis of the image) to world. We obtain the value for the spectral axis from that world vector, and add it to the vector of frequencies. Then we convert that vector of frequencies to velocity. Then we convert it back to frequency. They better agree.

---

coordsys.parentname.html

coordsys.parentname - Function

1.1.3 Get parent image name.

Description

This function returns the parent image name for ‘coordsys’.

Arguments

Returns
string

---

coordsys.setparentname.html

coordsys.setparentname - Function

1.1.3 Set the parent image name (normally not needed by end-users)

Arguments

Inputs	
imagename	String named parent image
	allowed: string
	Default:

Returns
bool

---

imagepol-Tool.html

1.1.4 imagepol - Tool

Polarimetric analysis of images
Requires: image **Synopsis**

Description

Summary
An Imagepol `tool` provides specialized polarimetric analysis of images. Some of these things could be done with the Lattice Expression Language (LEL; see note223) but are more conveniently offered separately.

General
Before it can be used, the Imagepol tool must be attached to an image (CASA, FITS, and Miriad formats are supported) with a Stokes coordinate axis. What you can then do with your Imagepol `tool` depends on exactly which Stokes parameters you have in the image. You must have some combination of Stokes I, Q, U and V on the Stokes axis. These refer to total intensity, two components of linear polarization, and circular polarization, respectively. Therefore, if you ask for linear polarization and the image only has Stokes I and V, you will get an error.
The Imagepol `tool` functions generally return, by default, an on-the-fly Image `tool` as their output. In most cases, this is a "virtual" image. There are a range of different sorts of "virtual" images in CASA (see Image). But the Imagepol `tool functions` generally return reference Image `tools`. That is, these reference different pieces of the original image attached to the Imagepol `tool`, either directly, or as mathematical expressions (e.g. the polarized intensity). If you delete the attached image, you render your Imagepol `tool` and its outputs useless. If you wish, rather than return a virtual image `tool`, you can fill in the `outfile` argument of most Imagepol `tool` functions and write the image out to disk, associating the Image `tool` with the disk file.
In some of the functions, the standard deviation of the thermal noise is needed. This is for debiasing polarized intensity images or working out statistical error images. By default it is worked out for you from the attached image with outliers from the mean discarded. Since it is the thermal noise it is trying to find, it is worked out from the V, Q & U, and finally I data in that order of precedence. This is because Stokes V is much less likely to contain source signal than Stokes I. You can supply the noise level if you know it better. For example, for small images or images with few signal-free pixels, the theoretical estimate may be better.

Analysis and Display

Traditionally, when generating secondary and tertiary images (e.g. position angle, fractional polarization, rotation measure etc), one masks the output image according to some statistical test. For example, if the error in the output image is greater than some value, or the errors in the input images are greater than some value. Imagepol `tools` do not offer this kind of masking. It does provide you with the error images for the derived images. By using LEL when you analyze your images, you can mask the images however you want when you use them. That is, we defer the error interpretation as long as possible. Here is an example.

```
"""
#
print "\t----\t Tool level Ex 1 \t----"
potool = casac.homefinder.find_home_by_name('imagepolHome')
po = potool.create()
po.imagepoltestimage(outfile='stokes.image')  # Create test image
po.close()                       # Close so we can illustrate opening an image
po.open('stokes.image')          # Open image with Imagepol tool
lpa = po.linpolposang()          # Linearly polarized position angle image
lpaerr = po.sigmalinpolposang() # Error in linearly polarized position angle image
lpa.statistics();                # Get statistics on position angle image
#viewer(mask=lpaerr.name()+'<5')  # Display when p.a. error < 5 degrees
#
"""
```

Display is handled via the Viewer `tool`. It can display and overlay combinations of raster, contour and vector representations of your data.
It is common to display linear polarization data via vectors where the position angle of the vector is the position angle of the linear polarized radiation, and the amplitude of the vector is proportional to either the total polarized intensity or fractional polarized intensity.
The data source of a vector display is either a Complex or a Float image. If it is a Complex image (e.g. the complex linear polarization $Q + iU$) then both the amplitude and the phase (position angle) are available. If it is just a Float image, then it is assumed to be the position angle and an amplitude of unity will be provided. The angular units are given by image brightness units which you can set with function setbrightnessunits. If the units are not recognized as angular, degrees are assumed.
The position angle is measured positive North through East when you display a plane holding a celestial coordinate (the usual astronomical convention). For other axis/coordinate combinations, a positive position angle is measured from +x to +y in the absolute world coordinate frame.
The Imagepol `tool` can create Complex disk images for you via functions complexlinpol (complex linear polarization), complexfraclinpol (complex

fractional linear polarization) and makecomplex (takes amp/phase or real/imag). As well as these Complex images, you can also make Float images of the linearly polarized intensity, linearly polarized position angle, and the fractional linearly polarized intensity (see below).
Now, the Image `tool` cannot yet deal with Complex images (it will in the future). This means that you cannot currently do

```
"""
#
print "\t----\t Tool level Ex 2 \t----"
po.open('stokes.image')  # Open image with Imagepol tool
po.complexlinpol('clp')  # Make complex image of linear polarization disk file
try:
  print "Expect SEVERE error and Exception here"
  ia.open('clp')         # Error
except Exception, e:
  print "Expected exception occurred!"
po.close()
#
"""
```

which is a bit annoying presently. However, the Viewer `tool` is able to read Complex images so that you are able to display them ok.

```
"""
#
print "\t----\t Tool level Ex 3 \t----"
po.open('stokes.image')   # Open image with Imagepol tool
po.complexlinpol('clp2')  # Make complex image of linear polarization disk file
#viewer()                 # Start viewer to give access to Complex image
#
"""
```

If you wanted to make a vector map display you would select the appropriate image in the Viewer's data manager GUI, click 'Vector Map' on the right hand side and it would appear in the display. Note that the Viewer's Vector map display capability also offers you amplitude noise debiasing and the On-The-Fly mask.

Overview of Imagepol `tool` functions

- Access to the different Stokes subimages is via functions stokes, stokesi, stokesq, stokesu, and stokesv.
- Create the standard secondary and tertiary polarization images with
  - complexfraclinpol - complex fractional linear polarization

  - complexlinpol - complex linear polarization
  - makecomplex - make complex image from amp/phase or real/imag
  - pol - polarized quantities as specified
  - linpolint - linearly polarized intensity
  - linpolposang - linearly polarized position angle
  - totpolint - total polarized intensity
  - fraclinpol - fractional linear polarization
  - fractotpol - fractional total polarization
  - depolratio - linear depolarization ratio

- Create the standard secondary and tertiary polarization error images (simple propagation of Gaussian errors) with

  - sigma - best guess at thermal noise
  - sigmastokes - specified Stokes parameter
  - sigmastokesi - Stokes I
  - sigmastokesq - Stokes Q
  - sigmastokesu - Stokes U
  - sigmastokesv - Stokes V
  - sigmalinpolint - linearly polarized intensity
  - sigmalinpolposang - linearly polarized position angle
  - sigmatotpolint - total polarized intensity
  - sigmafraclinpol - fractional linear polarization
  - sigmafractotpol - fractional total polarization
  - sigmadepolratio - linear depolarization ratio

- You can compute Rotation Measure images via either the traditional techniques involving a number of different frequencies (regularly or irregularly spaced) with function rotationmeasure or via a new Fourier Technique for many regularly-spaced frequencies with function fourierrotationmeasure.

Methods

imagepoltestimage	Attach the Imagepol tool to a test image file
complexlinpol	Complex linear polarization
complexfraclinpol	Complex fractional linear polarization
depolratio	Linear depolarization ratio
close	Close the image tool
done	Close this Imagepol tool

fourierrotationmeasure	Find Rotation Measure (Fourier approach)
fraclinpol	Fractional linear polarization
fractotpol	Fractional total polarization
linpolint	Linearly polarized intensity
linpolposang	Linearly polarized position angle
makecomplex	Make a Complex image
open	Open a new image with this imagepol tool
pol	Polarized quantities
rotationmeasure	Find Rotation Measure (traditional approach)
sigma	Find best guess at thermal noise
sigmadepolratio	Error in linear depolarization ratio
sigmafraclinpol	Error in fractional linear polarization
sigmafractotpol	Error in fractional total polarization
sigmalinpolint	Error in linearly polarized intensity
sigmalinpolposang	Error in linearly polarized position angle
sigmastokes	Find standard deviation of specified Stokes data
sigmastokesi	Find standard deviation of Stokes I data
sigmastokesq	Find standard deviation of Stokes Q data
sigmastokesu	Find standard deviation of Stokes U data
sigmastokesv	Find standard deviation of Stokes V data
sigmatotpolint	Error in total polarized intensity
stokes	Stokes
stokesi	Stokes I
stokesq	Stokes Q
stokesu	Stokes U
stokesv	Stokes V
summary	Summarise Imagepol tool
totpolint	Total polarized intensity

imagepol.imagepoltestimage.html

imagepol.imagepoltestimage - Function

1.1.4 Attach the Imagepol tool to a test image file

Description

This function can be used to generate a test image and then attach the Imagepol `tool` to it.
The test image is 4-dimensional (RA, DEC, Stokes and Frequency). The Stokes axis holds I,Q,U and V. The source is just a constant I (if you don't add noise all spatial pixels will be identical) and V. Q and U vary with frequency according to the specified Rotation Measure components (no attempt to handle bandwidth smearing within channels is made). The actual values of I,Q,U, and V are chosen arbitrarily otherwise (could be added as arguments if desired).
You can use this image, in particular, to explore the Rotation Measure algorithms in functions rotationmeasure and fourierrotationmeasure.
If you don't specify the Rotation Measure, then it is chosen for you so that there is no position angle ambiguity between adjacent channels (the value will be sent to the Logger).
The noise added to the image is specified as a fraction of the total intensity (constant). Gaussian noise with a standard deviation of `sigma *` I_{max} is then added to the image.

Arguments

Inputs		
outfile	Output image file name	
	allowed:	string
	Default:	imagepol.iquv
rm	Rotation Measure (rad/m/m). Default is auto no-ambiguity determine.	
	allowed:	doubleArray
	Default:	0.0
pa0	Position angle (degrees) at zero wavelength	
	allowed:	double
	Default:	0.0
sigma	Fractional noise level	
	allowed:	double
	Default:	0.01
nx	Shape of image in x direction	
	allowed:	int
	Default:	32
ny	Shape of image in y direction	
	allowed:	int
	Default:	32
nf	Shape of image in frequency direction	
	allowed:	int
	Default:	32
f0	Reference frequency (Hz)	
	allowed:	double
	Default:	1.4e9
bw	Bandwidth (Hz)	
	allowed:	double
	Default:	128.0e6

Returns
bool

Example

```
"""
#
print "\t----\t imagepoltestimage Ex 1 \t----"
po.imagepoltestimage(outfile='imagepoltestimage', rm=200)
po.rotationmeasure(rm='rm.out',rmmax=250)
```

```
ia.open('rm.out')
ia.statistics()
#viewer()
#
"""
```

---

imagepol.complexlinpol.html

imagepol.complexlinpol - Function

1.1.4 Complex linear polarization

Description

This function produces the complex linear polarization; $Q + iU$ and writes it to a disk image file.
The Image `tool` cannot yet handle Complex images. You must therefore write the Complex image to disk. The Viewer can display Complex images. Also the Table tool can access Complex images.

Arguments

Inputs		
outfile	Output image file name	
	allowed:	string
	Default:	

Returns
bool

Example

```
"""
#
print "\t----\t complexlinpol Ex 1 \t----"
po.open('stokes.image')
po.complexlinpol('cplx')
tb.open('cplx')
#tb.browse()
#
"""
```

imagepol.complexfraclinpol.html

imagepol.complexfraclinpol - Function

1.1.4 Complex fractional linear polarization

Description

This function produces the complex fractional linear polarization; $(Q + iU)/I$ and writes it to a disk image file.
The Image `tool` cannot yet handle Complex images. You must therefore write the Complex image to disk. The Viewer can display Complex images. Also the Table tool can access Complex images.

Arguments

Inputs		
outfile	Output image file name	
	allowed:	string
	Default:	

Returns
bool

Example

```
"""
#
print "\t----\t complexfraclinpol Ex 1 \t----"
po.open('stokes.image')
po.complexfraclinpol('cplx2')
tb.open('cplx2')
#tb.browse()
#
"""
```

imagepol.depolratio.html

imagepol.depolratio - Function

1.1.4 Linear depolarization ratio

Description

This function returns the linear depolarization ratio computed from two frequencies; this is the ratio of the fractional linear polarization at the two frequencies. Generally this is done when you have generated two images, each at a different frequency (continuum work). Thus if the fractional linear polarization images are $m_{\nu 1}$ and $m_{\nu 2}$ then the depolarization ratio is $m_{\nu 1}/m_{\nu 2}$. This function operates with two images; the first (at frequency $\nu 1$) is the one attached to your Imagepol `tool`. The second (at frequency $\nu 2$) is supplied via the argument `infile`, which is a String holding the name of the `image file`. In generating the depolarization ratio, you may optionally debias the linearly polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.
You can get the depolarization ratio error image with function sigmadepolratio.

Arguments

Inputs		
infile	Other image	
	allowed:	string
	Default:	
debias	Debias the linearly polarized intensity ?	
	allowed:	bool
	Default:	false
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0
sigma	Standard deviation of thermal noise. Default is auto determined.	
	allowed:	double
	Default:	-1
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	

Returns
image

Example

```
"""
#
#print "\t----\t depolratio Ex 1 \t----"
#po.open('stokes.4800')
#dpr = po.depolratio(infile='stokes.8300')        # m_4800 / m_8300
#edpr = po.sigmadepolratio(infile='stokes.8300');
#dpr.done()
#edpr.done()
#
"""
```

---

imagepol.close.html

imagepol.close - Function

1.1.4 Close the image tool

Description

This function closes the imagepol tool. This means that it detaches the tool from its `image file` (flushing all the changes first). The imagepol tool is "null" after this change (it is not destroyed) and calling any `tool function` other than open will result in an error.

Arguments

Returns
bool

Example

```
"""
#
print "\t----\t close Ex 1 \t----"
# First create image and attach it to imagepol tool
po.imagepoltestimage('myimagepol')
po.close()              # Detaches image from Imagepol tool
print "!!!EXPECT ERROR HERE!!!"
po.summary()            # No image so this results in an error.
po.open('myimagepol')   # Image is reattached
po.summary()            # No error
po.close()
#
"""
```

imagepol.done.html

imagepol.done - Function

1.1.4 Close this Imagepol tool

Description

This function is the same as close().

Arguments

Returns
bool

Example

```
"""
#
print "\t----\t done Ex 1 \t----"
po.open('myimagepol')
po.done()             # Detaches image from Imagepol tool
print "!!!EXPECT ERROR HERE!!!"
po.summary()          # No image so this results in an error.
po.open('myimagepol') # Image is reattached
po.summary()          # No error
po.done()
#
"""
```

imagepol.fourierrotationmeasure.html

imagepol.fourierrotationmeasure - Function

1.1.4 Find Rotation Measure (Fourier approach)

Description

This function will only work if you attach the Imagepol `tool` (using open) to an image containing Stokes Q and U, and a regular frequency axis. It Fourier transforms the complex linear polarization (Q+iU) over the spectral axis to the rotation measure axis. Thus, if your input image contained RA/DEC/Stokes/Frequency, the output image would be RA/DEC/RotationMeasure. The Rotation Measure axis has as many pixels as the spectral axis.
This method enables you to see the polarization as a function of Rotation Meausure. Its main use is when searching for large RMs. See Killeen, Fluke, Zhao and Ekers (1999, preprint) for a description of this method (or http://www.atnf.csiro.au/`$\sim$`nkilleen/rm.ps) and its advantages over the traditional method (rotationmeasure) of extracting the Rotation Measure.
Although you can write out the complex polarization image with the argument `complex`, you can't do much with it because Image `tools` cannot handle complex images. Hence you can also write out the complex linear polarization image in any or all of the other forms.
The argument `zerolag0` allows you to force the zero lag (or central bin) of the Rotation Measure spectrum to zero (effectively by subtracting the mean of Q and U from the Q and U images). This may avoid Gibbs phenomena from any strong low Rotation Measure signal which would normally fall into the central bin.

Arguments

Inputs	
complex	Output complex linear polarization image file name. Default is unset. allowed: string Default:
amp	Output linear polarization amplitude image file name Default is unset. allowed: string Default:
pa	Output linear polarization position angle (degrees) image file name Default is unset. allowed: string Default:
real	Output linear polarization real image file name Default is unset. allowed: string Default:
imag	Output linear polarization imaginary angle image file name Default is unset. allowed: string Default:
zerolag0	Force zero lag to 0 ? allowed: bool Default: false

Returns
bool

Example

```
"""
#
print "\t----\t fourierrotationmeasure Ex 1 \t----"
po.imagepoltestimage(outfile='iquv.im', rm=[5.0e5,1e6], nx=8, ny=8, nf=512,
                     f0=1.4e9, bw=8e6)
po.fourierrotationmeasure(amp='amp')
ia.open('amp')
ia.statistics()
#viewer()                      # And reorder to put RM along X-axis
#
```

"""

---

imagepol.fraclinpol.html

imagepol.fraclinpol - Function

1.1.4 Fractional linear polarization

Description

This function returns the fractional linear polarization; $\sqrt{(Q^2 + U^2)}/I$.
You may optionally debias the polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Arguments

Inputs		
debias	Debias the linearly polarized intensity ?	
	allowed:	bool
	Default:	false
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0
sigma	Standard deviation of thermal noise. Default is auto determined.	
	allowed:	double
	Default:	-1
outfile	Output image file name Default is unset.	
	allowed:	string
	Default:	

Returns
image

Example

"""

```
#
print "\t----\t fraclinpol Ex 1 \t----"
po.open('stokes.image')
flp = po.fraclinpol()
flp.summary()
flp.done()
#
"""
```

---

imagepol.fractotpol.html

imagepol.fractotpol - Function

1.1.4 Fractional total polarization

Description

This function returns the fractional linear polarization; $\sqrt{(Q^2+U^2+V^2)}/I$.
You may optionally debias the polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.
If your image contains only Q and U, or only V, then just those components contribute to the total polarized intensity.

Arguments

Inputs		
debias	Debias the total polarized intensity ?	
	allowed:	bool
	Default:	false
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0
sigma	Standard deviation of thermal noise. Default is auto determined.	
	allowed:	double
	Default:	-1
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	

Returns
image

Example

```
"""
#
print "\t----\t fractotpol Ex 1 \t----"
po.open('stokes.image')
ftp = po.fractotpol()
ftp.statistics()
ftp.done()
#
"""
```

---

imagepol.linpolint.html

imagepol.linpolint - Function

1.1.4 Linearly polarized intensity

Description

This function returns the linearly polarized intensity; $\sqrt{(Q^2 + U^2)}$.
You may optionally debias the polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Arguments

Inputs		
debias	Debias the linearly polarized intensity ?	
	allowed:	bool
	Default:	false
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0
sigma	Standard deviation of thermal noise. Default is auto determined.	
	allowed:	double
	Default:	-1
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	

Returns
image

Example

"""

```
#
print "\t----\t linpolint Ex 1 \t----"
po.open('stokes.image')
lpi = po.linpolint()
lpi.statistics()
lpi.done()
#
"""
```

---

imagepol.linpolposang.html

imagepol.linpolposang - Function

1.1.4 Linearly polarized position angle

Description

This function returns the linearly polarized position angle image ($0.5 \tan^{-1}(U/Q)$) in degrees.

Arguments

Inputs		
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	

Returns

image

Example

```
"""
#
print "\t----\t linpolposang Ex 1 \t----"
po.open('stokes.image')
lppa = po.linpolposang()
lppa.statistics()
lppa.done()
#
"""
```

imagepol.makecomplex.html

imagepol.makecomplex - Function

1.1.4 Make a Complex image

Description

This function generates a Complex `image file` from either a real and imaginary, or an amplitude and phase pair of images. If you give a linear position angle image for the phase, it will be multipled by two before the real and imaginary parts are formed.

Arguments

Inputs	
complex	Output complex image file name. Must be specified. allowed: string Default:
real	Input real image file name. Default is unset. allowed: string Default:
imag	Input imaginary image file name. Default is unset. allowed: string Default:
amp	Input amplitude image file name. Default is unset. allowed: string Default:
phase	Input phase image file name. Default is unset. allowed: string Default:

Returns

bool

Example

```
"""
#
print "\t----\t makecomplex Ex 1 \t----"
po.open('stokes.image')
po.complexlinpol('qu.cplx1')
q = po.stokesq()
u = po.stokesu()
q2 = q.subimage('q',overwrite=true)
u2 = u.subimage('u',overwrite=true)
po.makecomplex('qu.cplx2', real='q', imag='u')
po.close()
#
"""
```

In this example we make two complex linear polarization images which should be identical.

---

imagepol.open.html

imagepol.open - Function

1.1.4 Open a new image with this imagepol tool

Description

Before polarimetric analysis can commence, an `image file` must be attached to the imagepol tool using the open function. Also, use this function when you are finished analyzing the current `image file` and want to attach to another one. This function detaches the `image tool` from the current `image file`, if one exists, and reattaches it (opens) to the new `image file`.
The input image file may be in native CASA, FITS, or Miriad format. Look here for more information on foreign images.
The input image must have a Stokes axis. The exact collection of Stokes that the image has, determines what the Imagepol tool can compute. Stokes I, Q, U, and V refer to total intensity, two components of linear polarization, and circular polatization, respectively. Therefore, if you ask for linear polarization and the image only has Stokes I and V, you will get an error.
The input image may contain data at many frequencies. For example, the image may be a 4D image with axes RA, DEC, Stokes and Frequency (order not important) where the Frequency axis is regularly sampled. However, the image may also contain many frequencies at irregular intervals. Such an image may be created with the Image tool function imageconcat. It enables you to concatenate images along an axis, and it allows irregular coordinate values along that axis.

Arguments

Inputs		
image	image file name or image record (generated by ia.torecord())	
	allowed:	any
	Default:	variant

Returns
bool

Example

```
"""
#
print "\t----\t open Ex 1 \t----"
po.open('stokes.image')
po.close()
#
"""

The {\stff open} function first closes the old \imagefile\ if one exists.
```

---

imagepol.pol.html

imagepol.pol - Function

1.1.4 Polarized quantities

Description

This function just packages the other specific polarization functions into one where you specify an operation with the argument `which` (can be useful for scripts). This argument can take the values:

- 'lpi' - linearly polarized intensity (function linpolint)
- 'tpi' - total polarized intensity (function totpolint)
- 'lppa' linearly polarized position angle (function linpolposang)
- 'flp' - fractional linear polarization (function fraclinpol)
- 'ftp' - fractional total polarized intensity (function fractotpol)

Arguments

Inputs		
which	Specify operation. One of 'lpi', 'tpi', 'lppa', 'flp', 'ftp' (case insensitive)	
	allowed:	string
	Default:	
debias	Debias the polarized intensity ?	
	allowed:	bool
	Default:	false
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0
sigma	Standard deviation of thermal noise. Default is auto determined.	
	allowed:	double
	Default:	-1
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	

Returns
image

Example

```
"""
#
print "\t----\t pol Ex 1 \t----"
po.open('stokes.image')
lpi = po.pol('lpi')
lpi.statistics()
lpi.done()
#
"""
```

---

imagepol.rotationmeasure.html

imagepol.rotationmeasure - Function

1.1.4 Find Rotation Measure (traditional approach)

Description

This function generates the rotation measure image from a collection of different frequencies. It will only work if the Imagepol `tool` is attached to an image containing Stokes Q and U, and a frequency axis (regular or irregular) with at least 2 pixels. It will work out the position angle images for you.
See also the fourierrotationmeasure function for a new Fourier-based approach.
Rotation Measure algorithms that work robustly are not common. The main problem is in trying to account for the $n - \pi$ ambiguity (see Leahy et al, Astronomy & Astrophysics, 156, 234 or Killeen et al; http://www.atnf.csiro.au/`$\sim$`nkilleen/rm.ps).
The algorithm that this function uses is that of Leahy et al. (see Appendix A.1). But as in all these algorithms, the basic process is that for each spatial pixel, a vector of position angles (i.e. at the different frequencies) is fit to determine the rotation measure and the position angle at zero wavelength (and their errors). An image containing the number of $n - \pi$ turns that were added to the data at each spatial pixel and for which the best fit was found can be written. The reduced chi-squared image for the fits can also be written. Note that no assessment of curvature (i.e. deviation from the simple linear position angle - λ^2 functional form) is made.
Any combination of output images can be written.
The argument `sigma` gives the thermal noise in Stokes Q and U. By default it is determined automatically using the image data. But if it proves to be inaccurate (maybe not many signal-free pixels), it may be specified. This is used for calculating the error in the position angles (propagation of Gaussian errors).
The argument `maxpaerr` specifies the maximum allowable error in the position angle that is acceptable. The default is an infinite value. From the standard propagation of errors, the error in the linearly polarized position angle is determined from the Stokes Q and U images (at each spatial pixel for each frequency). At each spatial pixel we do a fit to the position angle vector (i.e. at the different frequencies) to determine the rotation measure. If the position angle error for any pixel in the vector exceeds the specified value, it is dropped from the fit. The process generates an error for the fit and this is used to compute the errors in the output images.
Note that `maxpaerr` is *not* used to specify that any pixel for which the output position angle error exceeds this value should be masked out.

The argument `rmfg` is used to specify a foreground RM value. For example, you may know the mean RM in some direction out of the Galaxy, then including this can aid the algorithm by reducing ambiguity.
The argument `rmmax` specifies the maximum absolute RM that should be solved for. This quite an important parameter. If you leave it at the default, zero, no ambiguity handling will be used. So some apriori information should be supplied; this is the basic problem with rotation measure algorithms.

Arguments

Inputs	
rm	Output Rotation Measure image file name. Default is unset. allowed: string Default:
rmerr	Output Rotation Measure error image file name. Default is unset. allowed: string Default:
pa0	Output position angle (degrees) at zero wavelength image file name. Default is unset. allowed: string Default:
pa0err	Output position angle (degrees) at zero wavelength error image file name. Default is unset. allowed: string Default:
nturns	Output number of turns image file name. Default is unset. allowed: string Default:
chisq	Output reduced chi squared image file name. Default is unset. allowed: string Default:
sigma	Estimate of the thermal noise. Default is auto estimate. allowed: double Default: -1
rmfg	Foreground Rotation Measure (rad/m/m) to subtract. allowed: double Default: 0.0
rmmax	Maximum rotation measure (rad/m/m) to solve for. IMPORTANT TO SPECIFY. allowed: double Default: 0.0
maxpaerr	Maximum input position angle error (degrees) to allow allowed: double Default: 1e30
plotter	Name of plotter. Default is none. allowed: string Default:
nx	Number of plots in x direction allowed: int Default: 5
ny	Number of plots in y direction allowed: int Default: 5

Returns
bool

Example

```
"""
#
print "\t----\t rotationmeasure Ex 1 \t----"
#im = ia.imageconcat(outfile='stokes.image',
#                    infiles="im.f1 im.f2 im.f3 im.f4 im.f5", axis=4)
po.open('stokes.image')
ok = po.rotationmeasure(rm='rm', rmerr='rmerr', rmmax=800, maxpaerr=10)
#
"""
```

```
Say we have 5 images, each with axes RA, DEC, Stokes, and Frequency in
that order.  We use the Image \tool\ to concatenate these images
along the frequency axis - you have ordered them in increasing or
decreasing frequency order.  We then compute the Rotation Measure
and Rotation Measure error images with the traditional method and
write them out to disk.
```

---

imagepol.sigma.html

imagepol.sigma - Function

1.1.4 Find best guess at thermal noise

Description

This function returns the standard deviation from V, Q&U or I in that order of precedence. It is attempting to give you the best estimate of the thermal noise it can from the data. Outliers from the mean are clipped at the specified level.

Arguments

Inputs		
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0

Returns

double

Example

```
"""
#
print "\t----\t sigma Ex 1 \t----"
po.open('stokes.image')
sigma = po.sigma()
print "sigma=", sigma
#
"""
```

imagepol.sigmadepolratio.html

imagepol.sigmadepolratio - Function

1.1.4 Error in linear depolarization ratio

Description

This function returns the error in the linear depolarization ratio computed from two frequencies; this is the ratio of the fractional linear polarization at the two frequencies. Generally this is done when you have generated two images, each at a different frequency (continuum work). Thus if the fractional linear polarzation images are $m1$ and $m2$ then the depolarization ratio is $m1/m2$.
This function operates with two images; the first is attached to the Imagepol `tool`. The second is supplied via the argument `infile`, which is a String holding the name of the `image file`.
In generating the depolarization ratio, and hence its error, you may optionally debias the linearly polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.
You can get the depolarization ratio image with function depolratio.

Arguments

Inputs		
infile	Other image. Required input.	
	allowed:	string
	Default:	
debias	Debias the linearly polarized intensity ?	
	allowed:	bool
	Default:	false
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0
sigma	Standard deviation of thermal noise. Default is auto determined.	
	allowed:	double
	Default:	-1
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	

Returns
image

Example

```
"""
#
#print "\t----\t sigmadepolratio Ex 1 \t----"
#po.open('stokes.4800')
#dpr = po.depolratio('stokes.8300')
#edpr = po.sigmadepolratio('stokes.8300');
#dpr.done()
#edpr.done()
#
"""
```

---

imagepol.sigmafraclinpol.html

imagepol.sigmafraclinpol - Function

1.1.4 Error in fractional linear polarization

Description

This function returns the error (standard deviation) of the fractional linear polarization. This result comes from standard propagation of errors. The result is an on-the-fly Image tool as the error is signal-to-noise ratio dependent. This function requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Arguments

Inputs		
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0
sigma	Standard deviation of thermal noise. Default is auto determined.	
	allowed:	double
	Default:	-1
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	

Returns
image

Example

```
"""
#
```

```
print "\t----\t sigmafraclinpol Ex 1 \t----"
po.open('stokes.image')
sigflp = po.sigmafraclinpol()
sigflp.statistics()
sigflp.done()           # free up resources
#
"""
```

---

imagepol.sigmafractotpol.html

imagepol.sigmafractotpol - Function

1.1.4 Error in fractional total polarization

Description

This function returns the error (standard deviation) of the fractional total polarization. This result comes from standard propagation of errors. The result is an on-the-fly Image tool as the error is signal-to-noise ratio dependent. This function requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Arguments

Inputs	
clip	Clip level for auto-sigma determination
	allowed: double
	Default: 10.0
sigma	Standard deviation of thermal noise. Default is auto determined.
	allowed: double
	Default: -1
outfile	Output image file name. Default is unset.
	allowed: string
	Default:

Returns
image

Example

```
"""
#
```

```
print "\t----\t sigmafractotpol Ex 1 \t----"
po.open('stokes.image')
sigftp = po.sigmafractotpol()
sigftp.statistics()
sigftp.done()
#
"""
```

---

imagepol.sigmalinpolint.html

imagepol.sigmalinpolint - Function

1.1.4 Error in linearly polarized intensity

Description

This function returns the error (standard deviation) of the linearly polarized intensity; $\sqrt{(Q^2 + U^2)}$. This result comes from standard propagation of statistical errors. The result is a float as the error is not signal-to-noise ratio dependent
This function requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Arguments

Inputs		
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0
sigma	Standard deviation of thermal noise. Default is auto determined.	
	allowed:	double
	Default:	-1
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	

Returns
double

Example

"""

```
#
print "\t----\t sigmalinpolint Ex 1 \t----"
po.open('stokes.image')
siglpi = po.sigmalinpolint()
print "siglpi=", siglpi
#
"""
```

---

imagepol.sigmalinpolposang.html

imagepol.sigmalinpolposang - Function

1.1.4 Error in linearly polarized position angle

Description

This function returns the error (standard deviation) of the linearly polarized position angle ($0.5 \tan^{-1}(U/Q)\sqrt{(Q^2+U^2)}$) in degrees. This result comes from standard propagation of errors. The result is an on-the-fly Image tool as the error is signal-to-noise ratio dependent.
This function requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Arguments

Inputs		
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0
sigma	Standard deviation of thermal noise. Default is auto determined.	
	allowed:	double
	Default:	-1
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	

Returns
image

Example

"""

```
#
print "\t----\t sigmalinpolposang Ex 1 \t----"
po.open('stokes.image')
siglppa = po.sigmalinpolposang()
siglppa.statistics()
siglppa.done()
#
"""
```

---

imagepol.sigmastokes.html

imagepol.sigmastokes - Function

1.1.4 Find standard deviation of specified Stokes data

Description

This function returns the standard deviation of the noise for the specified Stokes. Outliers from the mean are clipped at the specified level.

Arguments

Inputs		
which	Must specify Stokes parameter. One of 'I', 'Q', 'U', 'V' (case insensitive)	
	allowed:	string
	Default:	
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0

Returns

double

Example

```
"""
#
print "\t----\t sigmastokes Ex 1 \t----"
po.open('stokes.image')
sigq = po.sigmastokes('q', 10.0)
print "sigq=", sigq
#
"""
```

imagepol.sigmastokesi.html

imagepol.sigmastokesi - Function

1.1.4 Find standard deviation of Stokes I data

Description

This function returns the standard deviation of the noise for the Stokes I data. Outliers from the mean are clipped at the specified level.

Arguments

Inputs		
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0

Returns

double

Example

```
"""
#
print "\t----\t sigmastokesi Ex 1 \t----"
po.open('stokes.image')
sigi = po.sigmastokesi(10.0)
print "sigi=", sigi
#
"""
```

imagepol.sigmastokesq.html

imagepol.sigmastokesq - Function

1.1.4 Find standard deviation of Stokes Q data

Description

This function returns the standard deviation of the noise for the Stokes Q data. Outliers from the mean are clipped at the specified level.

Arguments

Inputs		
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0

Returns

double

Example

```
"""
#
print "\t----\t sigmastokesq Ex 1 \t----"
po.open('stokes.image')
sigq = po.sigmastokesq(10.0)
print "sigq=", sigq
#
"""
```

imagepol.sigmastokesu.html

imagepol.sigmastokesu - Function

1.1.4 Find standard deviation of Stokes U data

Description

This function returns the standard deviation of the noise for the Stokes U data. Outliers from the mean are clipped at the specified level.

Arguments

Inputs		
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0

Returns

double

Example

```
"""
#
print "\t----\t sigmastokesu Ex 1 \t----"
po.open('stokes.image')
sigu = po.sigmastokesu(10.0)
print "sigu=", sigu
#
"""
```

imagepol.sigmastokesv.html

imagepol.sigmastokesv - Function

1.1.4 Find standard deviation of Stokes V data

Description

This function returns the standard deviation of the noise for the Stokes V data. Outliers from the mean are clipped at the specified level.

Arguments

Inputs		
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0

Returns

double

Example

```
"""
#
print "\t----\t sigmastokesv Ex 1 \t----"
po.open('stokes.image')
sigv = po.sigmastokesv(10.0)
print "sigv=", sigv
#
"""
```

imagepol.sigmatotpolint.html

imagepol.sigmatotpolint - Function

1.1.4 Error in total polarized intensity

Description

This function returns the error (standard deviation) of the total polarized intensity; $\sqrt{(Q^2 + U^2 + V^2)}$. This result comes from standard propagation of statistical errors. The result is a float as the error is not signal-to-noise ratio dependent
This function requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Arguments

Inputs		
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0
sigma	Standard deviation of thermal noise. Default is auto determined.	
	allowed:	double
	Default:	-1

Returns
double

Example

```
"""
#
print "\t----\t sigmastotpolint Ex 1 \t----"
po.open('stokes.image')
sigtpi = po.sigmatotpolint()
```

```
print "sigtpi=", sigtpi
#
"""
```

---

imagepol.stokes.html

imagepol.stokes - Function

1.1.4 Stokes

Description

This function returns an on-the-fly image tool containing the specified Stokes only. This interface can be useful for scripts.

Arguments

Inputs		
which	Must specify Stokes. One of 'I', 'Q', 'U', 'V' (case insensitive)	
	allowed:	string
	Default:	
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	

Returns

image

Example

```
"""
#
print "\t----\t stokes Ex 1 \t----"
po.open('stokes.image')
q = po.stokes('q')
q.statistics()
q.done()
#
"""
```

imagepol.stokesi.html

imagepol.stokesi - Function

1.1.4 Stokes I

Description

This function returns an on-the-fly image tool containing Stokes I only.

Arguments

Inputs		
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	

Returns

image

Example

```
"""
#
print "\t----\t stokesi Ex 1 \t----"
po.open('stokes.image')
i = po.stokesi()
i.statistics()
i.done()
#
"""
```

imagepol.stokesq.html

imagepol.stokesq - Function

1.1.4 Stokes Q

Description

This function returns an on-the-fly image tool containing Stokes Q only.

Arguments

Inputs		
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	

Returns

image

Example

```
"""
#
print "\t----\t stokesq Ex 1 \t----"
po.open('stokes.image')
q = po.stokesq()
q.statistics()
q.done()
#
"""
```

imagepol.stokesu.html

imagepol.stokesu - Function

1.1.4 Stokes U

Description

This function returns an on-the-fly image tool containing Stokes U only.

Arguments

Inputs		
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	

Returns

image

Example

```
"""
#
print "\t----\t stokesu Ex 1 \t----"
po.open('stokes.image')
u = po.stokesu()
u.statistics()
u.done()
#
"""
```

imagepol.stokesv.html

imagepol.stokesv - Function

1.1.4 Stokes V

Description

This function returns an on-the-fly image tool containing Stokes V only.

Arguments

Inputs		
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	

Returns

image

Example

```
"""
#
print "\t----\t stokesv Ex 1 \t----"
po.open('stokes.image')
v = po.stokesv()
v.statistics()
v.done()
#
"""
```

imagepol.summary.html

imagepol.summary - Function

1.1.4 Summarise Imagepol tool

Description

This function just lists a summary of the Imagepol `tool` to the logger.
Currently it just summarizes the image to which the tool is attached.

Arguments

Returns
bool

Example

```
"""
#
print "\t----\t summary Ex 1 \t----"
po.open('stokes.image')
po.summary()
#
#Image name       : stokes.image
#Object name      :
#Image type       : PagedImage
#Image quantity   : Intensity
#Pixel mask(s)    : None
#Region(s)        : None
#
#Direction reference : J2000
#Spectral  reference : TOPO
#Velocity  type      : RADIO
```

```
#Rest frequency      : 1.4e+09 Hz
#Telescope           : UNKNOWN
#Observer            : UNKNOWN
#Date observation    : UNKNOWN
#
#Axis Coord Type      Name             Proj Shape Tile   Coord value at pixel    Coord incr
#-------------------------------------------------------------------------------------------
#0    0     Direction Right Ascension   SIN    32   32  00:00:00.000    16.00 -6.000000e+01
#1    0     Direction Declination       SIN    32   32 +00.00.00.000    16.00  6.000000e+01
#2    1     Stokes    Stokes                    4    4       I Q U V
#3    2     Spectral  Frequency                32   32       1.4e+09    16.00  4.000000e+06
#                     Velocity                                    0    16.00 -8.565499e+02
#
#
"""
```

---

imagepol.totpolint.html

imagepol.totpolint - Function

1.1.4 Total polarized intensity

Description

This function returns the total polarized intensity; $\sqrt{(Q^2+U^2+V^2)}$. If your image contains only Q and U, or only V, then just those components contribute to the total polarized intensity.
You may optionally debias the polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Arguments

Inputs		
debias	Debias the total polarized intensity ?	
	allowed:	bool
	Default:	false
clip	Clip level for auto-sigma determination	
	allowed:	double
	Default:	10.0
sigma	Standard deviation of thermal noised. Default is auto determined.	
	allowed:	double
	Default:	-1
outfile	Output image file name. Default is unset.	
	allowed:	string
	Default:	

Returns
image

Example

```
"""
#
print "\t----\t totpolint Ex 1 \t----"
po.open('stokes.image')
tpi = po.totpolint()
tpi.statistics()
tpi.done()
#
"""
```

---

components-Module.html

1.2 components - Module

Access and manipulate components

Description

This module contains functionality to manipulate model components
The available tools in this module are

- Componentlist - a tool for manupilating groups of components

Example

```
"""
#
print "\t----\t Module Ex 1 \t----"
pathname=os.environ.get("CASAPATH")
pathname=pathname.split()[0]
datapath=pathname+"/data/demo/Images/imagetestimage.fits"
ia.fromfits(outfile='testimage.im', infile=datapath, overwrite=T) # 1
hdr = ia.summary()                                                # 2
ia.statistics()                                                   # 3
ia.close()                                                        # 4
print "Last example!  Exiting..."
exit()
#
"""
```

componentlist-Tool.html

1.2.1 componentlist - Tool

A tool for the manipulation of groups of components
Requires:

Synopsis

Description

A componentlist is a tool that contains functions that manipulate components.
A component is a functional representation of the sky brightness and is
described in more detail in the overview of this module.
The simplest way to make a componentlist tool is to use the addcomponent
Thies creates a componentlist tool with a specified component but not
associated with any file.
The alternative is to use the open. This creates a componentlist tool by
reading the data form an CASA table.
The simplest way to add components to a list is to use the simulate function.
The components that are added to the list can then be edited using the
component editor, or any of the manipulate functions like setshape or setrefdir.

Methods

open	Construct an componentlist from the data in an CASA table
asciitocomponentlist	Create a componentlist from an ascii file **(Not implemented yet)**
concatenate	Append components from another componentlist.
fromrecord	make a componentlist tool from a record
torecord	convert componentlist to a record
remove	Remove a component from the list.
purge	Permanently delete removed components.
recover	Obtain removed components.
length	Find the number of components in the list.
indices	Return a vector of indices. **(Not implemented yet)**
sort	Sort the components in a list
isphysical	Check if a component is physically plausible
sample	Sample the flux of the list in a specified direction. **(Not implemented yet)**
rename	Give the list a name so it can save itself. use close to save to disk
simulate	Add some simulated components to the list
addcomponent	Add a component to the list
close	Save the componentlist to disk and reset its state.

edit	Start up the component editor gui **(Not implemented yet)**
done	Delete the componentlist tool
select	Mark components in the list
deselect	Unmark components in the list
selected	Determine which components are selected
getlabel	Get the label of the specified component
setlabel	Set the label of the specified components
getfluxvalue	Get the flux value of the specified component
getfluxunit	Get the flux unit of the specified component
getfluxpol	Get the polarization representation for the flux of the specified component **(Not imp**
getfluxerror	Get the error in the flux of the specified component
setflux	Set the flux of the specified components
convertfluxunit	Change (convert) the flux units of the specified components
convertfluxpol	Change (convert) the polarization representation of the specified components
getrefdir	Return the reference direction
getrefdirra	Get the RA of the reference direction. **(Not implemented not)**
getrefdirdec	Get the declination of the reference direction.**(Not implemented yet)**
getrefdirframe	Get the reference frame of the reference direction.
setrefdir	Set the reference direction
setrefdirframe	Set the reference frame for the direction
convertrefdir	Convert the reference direction to a new frame
shapetype	Returns the shape type of the component
getshape	Return the shape parameters the component
setshape	Change the shape of the component
convertshape	Change the units of the shape parameters **(Not implemented yet)**
spectrumtype	Returns the spectral shape of the component
getspectrum	Return the spectral parameters the component
setstokesspectrum	Change the spectrum of the component
setspectrum	Change the spectrum of the component
getfreq	Get the reference frequency **(Not implemented yet)**
getfreqvalue	Get the reference frequency value **(Not implemeted yet)**
getfrequnit	Get the reference frequency unit **(Not implemeted yet)**
getfreqframe	Get the reference frequency frame **(Not implemeted yet)**
setfreq	Set the reference frequency
setfreqframe	Set the reference frame for the frequency
convertfrequnit	Convert the reference frequency to a new unit
getcomponent	Extract a component from the list.
add	Add a component to the list.
replace	Replace components in the list. **(Not implemented yet)**
summarize	Summarize the specified component to the logger
iscomponentlist	Is the argument a componentlist tool? **(Not implemented yet)**

componentlist.open.html

componentlist.open - Function

1.2.1 Construct an componentlist from the data in an CASA table

Description

Use this constructor to construct a componentlist tool by reading the data from an CASA table. To ensure that this table contains all the necessary columns and to allow the table format to be enhanced in the future, it is highly recommended that the table be created using a componentlist tool[1]. The table that contains the componentlist may be opened read-only by setting the readonly flag to True. When this is done some of the functions in the componentlist tool cannot be used. These include the "set", "convert", "remove", "replace", "purge", "recover", and "sort" functions.

Arguments

Inputs		
filename	The filename of the table	
	allowed:	string
	Default:	
nomodify	Should the table be opened read only	
	allowed:	boolean
	Default:	false
log	Send a message to the logger	
	allowed:	boolean
	Default:	true

Returns

bool

Example

[1] To be more precise the table should have been created using the functions in the ComponentList class (C++).

```
cl.open('crux.cl');
```

---

componentlist.asciitocomponentlist.html

componentlist.asciitocomponentlist - Function

1.2.1 Create a componentlist from an ascii file **(Not implemented yet)**

Description

This constructor allows conversion of a number of ascii-file-based formats to componentlists. The formats currently supported are the AIPS star file format, the Caltech model format, and the WENSS, FIRST, and NVSS surveys.

AIPS ST file The AIPS star file describes positions of "stars" that one might wish to plot overlaid on some other display. It does not contain source strength information. In the AIPS help file for STARS, the format of the star file is described as follows:

```
    The text file contains one line per star and each line has
up to 7 logical columns containing, in order:
   1. X position (Right Ascension 00 00 00.00)
   2. Y position (Declination +/-00 00 00.0)
   3. Major axis (Full width in arc seconds on sky)
   4. Minor axis (Full width in arc seconds on sky)
   5. Position Angle (E of N, degrees)
   6. Star Type  (-1 to 20, integer )
   7. Star label (up to 24 character string)
If X and Y are not RA-DEC or DEC-RA, then the logical columns
are also 7 actual columns and the units are in AIPS standard
units (e.g. degrees, m/s etc. ).  In this case the position
angle should be given as 0.0, the major axis is the width in the
Y coordinate and the minor axis is the width in the X
coordinate.  For RA and DEC positions, the sexagesimal notation
is used (e.g. HH MM SS.SSS -DD MM SS.S) for the positions and
arc seconds on the sky are used for the Deltas.
The last 5 columns are not required.  If the last 5 columns are
not given, a value of 1 cell is assumed for the deltas.
If the position angle is not included, the default is 0 degrees.
If the star type is not included, the default type is a cross.
The default is no label string.
There are currently 22 different types of star marks.
 < 0: No Mark, only the star label is printed
   0: Cross             10: Five pointed star
   1: Ellipse           11: Star of David
```

```
   2: Box               12: Seven pointed star
   3: Triangle          13: Eight pointed star
   4: Diamond           14: Nine  pointed star
   5: Pentagon          15: Ten   pointed star
   6: Hexagon           16: 11    pointed star
   7: Septagon          17: 12    pointed star
   8: Octagon           18: 13    pointed star
   9: Nine-gon          19: 14    pointed star
  20: Cross with gap   >20: Ellipse
The Box (type=2) is different from the diamond in that the
star size is the half height and width of the box dimensions.
The Box and the Null (<0) are labeled at RA and Dec plus Delta
RA and Delta Dec.  The other marks are labeled at the right
edge of the of the Rotated RA axis.  The CROSS WITH GAP
(type=20) has the inner third of the cross removed so the
marked object is not over written.
```

For more information, see the AIPS help file for STARS.

Caveats:

- In AIPS, the following are supported: 1: 1900; 2: B1950; 3: J2000; 4: Galactic 5: OHLSSON Gal.; 6: VAN TULDER Galactic; 7: Super Galactic; if ¿ 1000 then year assumed. Currently, CASA can support 2 (refer='b1950'), 3 (refer='j2000'), 4 (refer='gal'), 7 (refer='supergal'). If you need any of the others, please contact us.

Caltech The Caltech package uses a format for specifying positions relative to an undefined position. In the documentation for *modelfit*, the format is described as follows:

```
     Model files are text files that can be typed or printed directly;
they  can  be  modified or created using the standard text editors.  A
model file consists of one line for each component of the model,  with
up to 7 numbers on each line (in free format):

     1.  Component flux density (Jy)

     2.  Distance of center of component  from  origin  (milliarcsec),
         "radius"

     3.  Position angle of center of component  with  respect  to  the
         origin (degrees, North through East), "theta"

     4.  Major axis of component (milliarcsec), "axis"

     5.  Axial ratio (minor/major, i.e.  < 1), "ratio"
```

6. Position angle of major axis (degrees, North through East), "phi"

7. Type:
   0 or 1: elliptical Gaussian (major axis is FWHM) or delta-function (major axis = 0)
   2: uniform elliptical disk (major axis is diameter)
   3: optically thin spheroid or tapered disk (major axis is diameter)
   4: elliptical ring (major axis is diameter)
   5: line (major axis is length)

For Gaussians and delta-functions, the Type can be omitted; for delta-functions, the major-axis, axial-ratio, position- angle and type can be omitted. (Not all the programs understand types 2-5.) The "origin" is an arbitrary phase-reference point. The maximum number of components varies from 600 to 10000, depending on the individual program.

Caveats:

- In CASA, directions in componentlist are currently absolute only. Hence one must specify the reference direction.

WENSS The Westerbork Northern Sky Survey (WENSS) is a low-frequency radio survey that covers the whole sky north of delta=30 degree at a wavelength of 92cm to a limiting flux density of approximately 18 mJy (5 sigma). This survey has a resolution of 54" x 54" cosec (delta) and a positional accuracy for strong sources of 1.5". The WSRT Northern Sky Survey catalog is available via a Web interface. Use this interface to search for sources, choose the plain text output, capture the output into a file and then convert.

FIRST Faint Images of the Radio Sky at Twenty-cm – is a project designed to produce the radio equivalent of the Palomar Observatory Sky Survey over 10,000 square degrees of the North and South Galactic Caps. Using the NRAO Very Large Array (VLA) and an automated mapping pipeline, they produce images with 1.8" pixels, a typical rms of 0.15 mJy, and a resolution of 5". At the 1 mJy source detection threshold, there are ∼90 sources per square degree, ∼35% of which have resolved structure on scales from 2-30". Go to the FIRST catalog search page, search for the sources that you want, cut out only the lines that include the sources, put in a file and then convert.

NVSS The NRAO VLA Sky Survey The NRAO VLA Sky Survey (NVSS) is a radio continuum survey covering the sky north of -40 deg

declination. A detailed description appears in the 1998 May issue of The Astronomical Journal (Condon, J. J., Cotton, W. D., Greisen, E. W., Yin, Q. F., Perley, R. A., Taylor, G. B., & Broderick, J. J. 1998, AJ, 115, 1693). Go to the NVSS catalog search page, search for the sources that you want, cut out only the lines that include the sources (you can leave in the alternating lines with error information), put in a file and then convert.

Caveats:

- The catalog contains upper limits on some scale sizes. We have chosen to represent these as actual sizes.

Arguments

Inputs	
filename	Name of output component list table allowed: string Default:
asciifile	Name of input ascii file allowed: string Default:
refer	Input reference frame allowed: string Default: B1950 J2000
format	Name of format (only ST supported) allowed: string Default: ST
direction	Direction measure (for relative coordinates) allowed: record Default:
spectrum	Default spectrum field, valid spectrum field [type="Constant", frequency=[type="frequency" , refer="LSR" , m0=[unit="GHz" , value=1.0]] allowed: record Default:
flux	Default flux field, valid flux field [value=[0.0, 0.0, 0.0, 0.0], unit='Jy', polarization="Stokes"] allowed: record Default:
log	Send a message to the logger allowed: bool Default: true

Returns
int

Example

```
mycl := asciitocomponentlist('sgra-stars.cl', 'sgra-stars.stfile', refer='j2000',
format='ST')
```

---

componentlist.concatenate.html

componentlist.concatenate - Function

1.2.1 Append components from another componentlist.

Description

The concatenate function copies the specified component(s), from the specified to list, to the end of the current list. The components are specified by numbering them from one to the length of the list. You cannot append components to a list that has been opened read only but the list you are copying from may be readonly.
You use a vector of indices to copy a number of components at once. By default all components are copied.

Arguments

Inputs		
list	list to copy from. Can be a componentlist record or a componentlist file name from disk	
	allowed:	any
	Default:	
which	which components to copy, -1 unset	
	allowed:	intArray
	Default:	-1
log	Send a message to the logger	
	allowed:	bool
	Default:	true

Returns
bool

Example

```
cl.addcomponent(flux=1.0, dir='J2000 19h00m00 -40d00m00')
cl.addcomponent(flux=2.0, dir='J2000 19h10m00 -40d00m00')
```

```
   cl.addcomponent(flux=3.0, dir='J2000 19h00m00 -40d00m00')
   cl2 = cltool();
   cl2.concatenate(cl.torecord(), [0,2]);
   cl.done()
   cl2.rename('part_list.cl');
   cl2.done()
```

We make a 3 component component list and
copies the first and third component to another a componentlist
that was initially empty. These components are then saved to the
table called part_list.cl.

```
  cl.close() ### make sure we start with empty componentlist
  cl.concatenate('crux.cl', [0,2]);
  cl.rename('crux-copy.cl');
  cl.done()
```

This example reads a componentlist from a casa table and
copies the first and third component to another a componentlist
that was initially empty. These components are then saved to the
table called crux-copy.cl.

---

componentlist.fromrecord.html

componentlist.fromrecord - Function

1.2.1 make a componentlist tool from a record

Description

This function allows the componentlist records that are returned by other functions (for e.g from imageanalysis tool) be converted to a tool to be manipulated or to be saved on disk

Arguments

Inputs		
record	a component list record	
	allowed:	record
	Default:	

Returns

bool

Example

```
cl2 = cltool()

cl2.fromrecord(ia.findsources())
cl2.rename('sourcesfound.cl')
cl2.done()
```

componentlist.torecord.html

componentlist.torecord - Function

1.2.1 convert componentlist to a record

Description

This function allows the componentlist to be converted to a record. Usually useful to pass to other functions in image analysis for e.g

Arguments

Inputs

Returns
record

Example

```
cl2=cltool()
cl.open('mycomp.cl')

cl2.fromrecord(ia.deconvolvecomponentlist(cl.torecord()))
cl2.rename('deconvolved_sources.cl')
cl2.done()
```

---

componentlist.remove.html

componentlist.remove - Function

1.2.1 Remove a component from the list.

Description

The remove function removes the specified component(s) from the list. Components are specified by numbering them from one to the length of the list. So removing component one will remove the first component. After using this function all the remaining components will be shuffled down so that component two becomes component one. You cannot remove components from a list that has been opened read only.
You can specify a vector of indices to remove a number of components at once. For example in a five element list removing elements [1,3,5] will result in a two element list, now indexed as elements one and two, containing what was previously the second and fourth components.
Components that have been deleted using this function are not lost. The recover function can be used to get them back unless the purge function has been executed. Then they are completely gone.

Arguments

Inputs		
which	indices of which component(s) to remove a vector containing unique integers between 0 and one less than the length of the list, -1 for all	
	allowed:	intArray
	Default:	-1
log	Send a message to the logger	
	allowed:	bool
	Default:	true

Returns
bool

Example

```
cl.open('crux.cl')
cl.remove(1)
```

---

componentlist.purge.html

componentlist.purge - Function

1.2.1 Permanently delete removed components.

Description

The remove function deletes components from the list but does not remove them from memory. They remain accessible and can be obtained with the recover function. The purge function frees up the memory occupied by the removed components. You cannot use the recover function to obtain the removed components after the purge function has been called.

Arguments

Returns
bool

Example

```
cl.open('crux.cl')
cl.remove(1)
cl.purge()
```

---

componentlist.recover.html

componentlist.recover - Function

1.2.1 Obtain removed components.

Description

The recover function appends components to the end of the list that have been deleted with the remove function. This does not include components that were removed before the purge function was last executed.

Arguments

Inputs		
log	Send a message to the logger	
	allowed:	bool
	Default:	true

Returns

bool

Example

```
cl.open('crux.cl')
cl.remove(1)
cl.recover()
```

---

componentlist.length.html

componentlist.length - Function

1.2.1 Find the number of components in the list.

Description

The length function returns a non-negative integer that indicates how many components the list currently contains.

Arguments

Returns
int

Example

```
cl.open('crux.cl')
n = cl.length()
```

---

componentlist.indices.html

componentlist.indices - Function

1.2.1 Return a vector of indices. **(Not implemented yet)**

Description

The indices function returns a vector of non-negative integers that can be used to index through the list. This vector always contains the integers starting at one and increasing sequentially to the length of the list. Its main use is in `for` loops as is illustrated in the example below.

Arguments

Returns
intArray

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
allcomp := cl.indices();
cl.convertfluxunit(allcomp, 'jy');
cl.convertfluxpol(allcomp, 'stokes');
totalflux := [0,0,0,0];
for (i in allcomp) {
  totalflux +:= cl.getfluxvalue(i);
}
```

componentlist.sort.html

componentlist.sort - Function

1.2.1 Sort the components in a list

Description

The sort function can sort all the components in a list using a variety of criteria. Currently the following criteria are available.

Flux Sorts the list so that the brightest components, as defined by the abs(I), are at the beginning of the list.

Position Sorts the list so that components that are closest to a reference position, which is currently fixed at (ra,dec) = $(0, 0)$, are at the beginning of the list.

Polarization Sorts the list so that components with the largest fractional polarization, $\frac{\sqrt{Q^2+U^2+V^2}}{I}$, are at the front of the list. Components where $I = 0$ are placed at the end of the list.

The parsing of the string containg the sorting criteria is case insensitive. You cannot sort a list that has been opened read only.

Arguments

Inputs		
criteria	a string containg the criteria to use to sort the list	
	allowed:	string
	Default:	Polarization
		Position
		Flux
log	Send a message to the logger	
	allowed:	bool
	Default:	true

Returns

bool

Example

```
cl.open('crux.cl')
cl.sort('Polarization')
```

---

componentlist.isphysical.html

componentlist.isphysical - Function

1.2.1 Check if a component is physically plausible

Description

The isphysical function is used to check if the specified components meet a number of criteria that must be true if the component could be used to model a physical process. These criteria are:

- $I >= \sqrt{Q^2 + U^2 + V^2}$
- That the flux, when represented using the Stokes representation, has a zero imaginary value.

The "Flux properties" section of the ComponentModels module documentation describes how it is possible to generate a component which has non-zero imaginary value in the Stokes representation.
It is possible to check a number of components at once by specifying the indicies of all the components. The returned value will only be True if all the specified components are physical.

Arguments

Inputs		
which	A vector of indices Indices must be between 0 and one less than the list length, inclusively	
	allowed:	intArray
	Default:	-1

Returns
bool

Example

```
cl2 = cltool()
```

```
cl2.simulate(2)
cl2.setflux(1, value=[10, 1+3j, 1-4j, 0], polarization="linear");
print cl2.isphysical([0,1])
```

---

componentlist.sample.html

componentlist.sample - Function

1.2.1 Sample the flux of the list in a specified direction. **(Not implemented yet)**

Description

The sample function returns a vector containing the flux in Janskys/pixel of all the components in the list, in the specified direction, at the specified frequency. The returned vector always contains four elements corresponding to the Stokes parameters I,∼Q,∼U,∼V.

Arguments

Inputs	
direction	The direction to sample any valid direction measure. A valid Direction measure or vector of string or string, e.g me.direction('J2000','19h30m00', '-20d00m00') or ['J2000','19h30m00', '-20d00m00'] or 'J2000 19h30m00 -20d00m00' allowed: any Default: J2000 00h00m00.00 90d00m00.0
pixellatsize	the x-size of the in pixels to use when sampling any quantity that has angular units. allowed: any Default: 0.0deg
pixellongsize	the y-size of the in pixels to use when sampling any quantity that has angular units. allowed: any Default: 0.0deg
frequency	The frequency to sample at Any frequency measure allowed: any Default: 1.4GHz

Returns
doubleArray

Example

```
include 'componentlist.g'
include 'measures.g'
cl := componentlist('crux.cl', readonly=T);
dir := dm.direction('J2000', '12h26m35.9', '-63d5m56');
pixelsize := dm.quantity('1arcsec');
flux := cl.sample(dir, pixelsize);
```

---

componentlist.rename.html

componentlist.rename - Function

1.2.1 Give the list a name so it can save itself. use close to save to disk

Description

The rename function is used to specify the name of the table associated with this componentlist.
When a componentlist is created using the emptycomponentlist constructor it is not associated with an casa table. So when the componentlist is removed from memory its contents are lost. But if a name is attached to the componentlist, using the rename function, then its contents are saved in a table with the specified name when the componentlist is closed
NOTE: that by just using `rename` the componentlist is not ensured to be on disk; to be sure use `close` after rename
If the componentlist is created using the componentlist constructor then this function will rename the table associated with the list to the user specified name. You cannot rename a componentlist that has been opened read only.

Arguments

Inputs		
filename	The filename of the table	
	allowed:	string
	Default:	
log	Send a message to the logger	
	allowed:	bool
	Default:	true

Returns
bool

Example

```
cl.simulate(1);
cl.setshape(0, 'gaussian', '35mas', '27mas', '-10d')
cl.setflux(0, [1.0, 0.2, 0.1, 0.01]);
cl.rename('smallblob.cl');
cl.close();

cl.open('smallblob.cl')
n=cl.length()
```

This example starts with an empty componentlist tool and then adds one component to it. The parameters of this component are then modified to change the shape and flux and the list saved in the casa table called 'smallblob.cl' The data is not written to disk until the list is closed, and when it is the componentlist is reset. So you need to reopen it if you want to interact with it.

---

componentlist.simulate.html

componentlist.simulate - Function

1.2.1 Add some simulated components to the list

Description

The simulate function adds simulated components to the list. The simulation criterion is very simple, all the components added are identical! They are point sources at the J2000 north pole with a flux in Stokes I of 1∼Jy, and zero in the other polarizations. The spectrum is constant. The 'set' functions (eg., setflux, setfreq) can be used to change these parameters to desired ones.
The howmany argument indicates how many components to append to the list.

Arguments

Inputs		
howmany	How many components to simulate, greater than zero	
	allowed:	int
	Default:	1
log	Send a message to the logger	
	allowed:	bool
	Default:	true

Returns

bool

Example

```
cl.simulate(2)
cl.setflux(1, [2.78, 0, 0, 0]);
cl.rename('test.cl');
cl.close();
```

```
This example creates a componentlist with two components.  The setflux function is used
modify the second component.  The list is then saved on disk.
I use short scripts like this a lot during testing.

I expect bad things will happen if you save the list to disk,
using the close
function, before having shut down the editor gui (using the done
button) or if you modify the same component using any of the set
functions while it is being modified by the gui.
```

---

componentlist.addcomponent.html

componentlist.addcomponent - Function

1.2.1 Add a component to the list

Description

The addcomponent function is a convenience function that ties together the simulate function, and the various set functions. This function adds a component to the end of the list. For a description of the arguments see the following functions.

flux, fluxunit, polarization See the setflux function.

ra, raunit, dec, decunit See the setrefdir function.

dirframe See the setrefdirframe function.

shape, majoraxis, minoraxis, positionangle See the setshape function.

freq A frequency quantity which is split into a value and units and passed to the setfreq function.

freqframe See the setfreq function.

spectrumtype, index The spectral index α such that flux density S as a function of frequency ν is given by the formula: $S \propto \nu^{\alpha}$

See the setspectrum function.

OR

setspectrum

label See the setlabel function.

Arguments

Inputs	
flux	The flux value. A vector with four real or complex numbers allowed: any Default:
fluxunit	The units of the flux. Any string with the same dimensions as the Jansky allowed: string Default: Jy
polarization	The polarization of the value field. “Stokes”, “linear” or “circular” allowed: string Default: Circular Linear Stokes
dir	The direction measure of the source, it can a be any direction measure from the measures tool or a string of the type ’J2000 10h30m00 -20d00m00.0’ or a vector of strings of the type [’J2000’, ’10:30:00.00’, ’-20.00.00.0’]. Basically the string or strings should have the direction frame and quantities for Ra and Dec allowed: any Default: J2000 00h00m00.0 90d00m00.0
shape	The new shape type. A string that is either ’point’, ’Gaussian’, ’disk’, or ’limbdarkeneddisk’ allowed: string Default: disk limbdarkeneddisk Gaussian point
majoraxis	The width (FWHM in the case of a Gaussian) of the larger axis. A quantity with angular units allowed: any Default: 2.0arcmin
minoraxis	The width (FWHM in the case of a Gaussian) of the smaller axis. A quantity with angular units allowed: any Default: 1.0arcmin
positionangle	The rotation of the axes with respect to the reference frame. A quantity with angular units allowed: any Default: 0.0deg
freq	The reference frequency. A quantity with units equivalent to the ’Hz’ and frame or a frequency measure, e.g [’TOPO’, ’1.6GHz’], or simply default frame (LSRK) ’1.6GHz’ allowed: any Default: LSRK 1.415GHz
spectrumtype	The spectrum type, a string that is either ’constant’ or ’spectral index’ allowed: string Default: spectral index constant
index	The spectral index allowed: double Default: 1.0

Returns
bool

---

componentlist.close.html

componentlist.close - Function

1.2.1 Save the componentlist to disk and reset its state.

Description

The close function resets the componentlist to its default state. In this state it contains no components and is not associated with any table.
This function flushes all the components in memory to disk if the componentlist is associated with a table. The table is then closed, and the contents of the list deleted.
If the list is not associated with a table its contents are still deleted and memory used by the list is released.

Arguments

Inputs		
log	Send a message to the logger	
	allowed:	bool
	Default:	true

Returns

bool

Example

```
See the example for the
rename function.
```

---

componentlist.edit.html

componentlist.edit - Function

1.2.1 Start up the component editor gui **(Not implemented yet)**

Description

The edit function starts up a graphical user interface which allows the user to view and manipulate individual components. The which argument specifies the component to edit.
The component being edited is copied into the componenteditor tool. Hence if you add or remove components or change the order of components in the list while the component is in the editor it will be put back in the wrong place! So do not manipulate the list while editting a component. It is also suggested you only edit one component at a time.

Arguments

Inputs		
which	An index specifying which component. An integer between 0 and one less than the list length	
	allowed:	int
	Default:	no default
log	Send a message to the logger	
	allowed:	bool
	Default:	true

Returns
bool

Example

```
See the example for the
simulate function.
```

componentlist.done.html

componentlist.done - Function

1.2.1 Delete the componentlist tool

Description

The done function frees up all the memory associated with a componentlist tool. After calling this function the componentlist tool cannot be used, either to manipulate the current list, or to open a new one. This function does not delete the disk file associated with a componentlist, but it will shut down the server process if there are no other componentlist tools being used.

Arguments

Returns
bool

Example

```
See the example for the
rename function.
```

---

componentlist.select.html

componentlist.select - Function

1.2.1 Mark components in the list

Description

The select function is used to mark the specified components as “selected”. This function will be used in conjunction with the planned graphical user interface. Other functions functions in the componentlist tool will behave no differently if a component is marked as “selected”.
Components are not selected when the list is initially read from disk or when a new component is added to the list using the simulate function.

Arguments

Inputs	
which	A vector of indices. Indices must be between 0 and one less than the list length, inclusively
	allowed: intArray
	Default:

Returns
bool

Example

```
cl.open('crux.cl')
cl.select([1,3])
```

componentlist.deselect.html

componentlist.deselect - Function

1.2.1 Unmark components in the list

Description

The deselect function is used to remove the "selected" mark from specified components in the list. This function wiil be used in conjunction with the planned graphical user interface and no other functions in the componentlist will behave differently if a component is marked as "selected" or not. Components are not selected when the list is initially read from disk or when a new component is added to the list using the simulate function. Deselecting a component that is already deselected is perfectly valid and results in no change.

Arguments

Inputs	
which	A vector of indices Indices must be between 0 and one less than the list length, inclusively
	allowed: intArray
	Default:

Returns

bool

Example

```
cl.open('crux.cl')
cl.select([1,3])
cl.deselect([2,3])
```

componentlist.selected.html

componentlist.selected - Function

1.2.1 Determine which components are selected

Description

The selected function is used to determine which components in a list are selected. It returns a vector with indices that indicate which components are selected. A zero length vector is returned if no components are selected. Components are marked as selected using the select function. This function will be used in conjunction with the graphical user interface and other functions in the componentlist tool will behave no differently if a component is marked as "selected" or not.

Arguments

Returns
intArray

Example

```
cl.open('crux.cl')
cl.select([1,3])
cl.deselect([2,3])
cl.selected()
```

componentlist.getlabel.html

componentlist.getlabel - Function

1.2.1 Get the label of the specified component

Description

The getlabel function returns the label associated with the specified component. The label is an arbitrary text string that can be used to tag a component.

Arguments

Inputs	
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int Default:

Returns

string

Example

```
cl.open('crux.cl')
cl.getlabel(1)
```

componentlist.setlabel.html

componentlist.setlabel - Function

1.2.1 Set the label of the specified components

Description

The setlabel function is used to reassign the label (an arbitrary text string) of the specified components to a new value.

Arguments

Inputs		
which	An index specifying the component to modify. An integer between 0 and one less than the list length, inclusively	
	allowed:	int
	Default:	no default
value	The label for the specified components	
	allowed:	string
	Default:	
log	Send a message to the logger	
	allowed:	bool
	Default:	true

Returns

bool

Example

```
cl.open('centarusA.cl')
cl.setlabel(1, 'Core')
```

componentlist.getfluxvalue.html

componentlist.getfluxvalue - Function

1.2.1 Get the flux value of the specified component

Description

The getfluxvalue function returns the value of the flux of the specified component using the current units and the current polarization representation. The functions getfluxunit & getfluxpol & can be used to get the units and polarization representation that corresponds to the supplied value.

Arguments

Inputs		
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively	
	allowed:	int
	Default:	no default

Returns

doubleArray

Example

```
cl.open('crux.cl');
flux = cl.getfluxvalue(1);
unit = cl.getfluxunit(1);
```

This example returns the values, units, polarization and error of the first component in the list.

componentlist.getfluxunit.html

componentlist.getfluxunit - Function

1.2.1 Get the flux unit of the specified component

Description

The getfluxunit function returns the units of the flux of the specified component. The actual values are obtained using the getfluxvalue function.

Arguments

Inputs	
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int Default:

Returns

string

Example

```
See the example for the
getfluxvalue function.
```

---

componentlist.getfluxpol.html

componentlist.getfluxpol - Function

1.2.1 Get the polarization representation for the flux of the specified component **(Not implmented yet)**

Description

The getfluxunit function returns the polarization representation of the flux of the specified component. The actual values are obtained using the getfluxvalue function.

Arguments

Inputs	
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int Default:

Returns

string

---

componentlist.getfluxerror.html

componentlist.getfluxerror - Function

1.2.1 Get the error in the flux of the specified component

Description

The getfluxerror function returns the error in the flux of the specified component using the current units and the current polarization representation. The functions getfluxvalue & getfluxunit & getfluxpol & can be used to get the value, units and polarization representation that corresponds to the specified error.
No error calculations are done by this tool. The error can be stored and retreived and if any of the parameters of the flux change the user is responsible for updating the errors.

Arguments

Inputs		
which	Index specifying which component. An integer between 0 and one less than the list length, inclusively	
	allowed:	int
	Default:	

Returns
doubleArray

---

componentlist.setflux.html

componentlist.setflux - Function

1.2.1 Set the flux of the specified components

Description

The setflux function is used to reassign the flux of the specified components to a new value. The flux value, unit and polarization can be specified and any number of components can be set to the new value. (Currently, the parameter, error is ignored.)

Arguments

Inputs	
which	A vector of indices specifying the components to modify. A vector with indices between 0 and one less than the list length, inclusively allowed: int Default:
value	The flux values for the specified components A vector with four real or complex numbers allowed: any Default:
unit	The units of the flux. Any string with the same dimensions as the Jansky allowed: string Default: Jy
polarization	The polarization of the value field allowed: string Default: circular linear Stokes
error	The error in the value field. A complex vector of length four. allowed: any Default:
log	Send a message to the logger allowed: bool Default: true

Returns
bool

Example

```
cl.open('crux.cl');
cl.setflux(0, [1,0,0,0], unit='jy',
   polarization='Stokes', error=[.3, 0, 0, 0])
```

---

componentlist.convertfluxunit.html

componentlist.convertfluxunit - Function

1.2.1 Change (convert) the flux units of the specified components

Description

The convertfluxunit function is used to convert the flux to another unit. The units *must* have the same dimensions as the Jansky.

Arguments

Inputs	
which	A vector of indices specifying the components to modify. A vector with indices between 0 and one less than the list length, inclusively allowed: int Default:
unit	The units of the flux. Any string with the same dimensions as the Jansky allowed: string Default: Jy

Returns

bool

Example

```
cl.open('crux.cl')
print cl.getfluxvalue(1)
cl.convertflux(1, 'WU')
print cl.getfluxvalue(1)
```

componentlist.convertfluxpol.html

componentlist.convertfluxpol - Function

1.2.1 Change (convert) the polarization representation of the specified components

Description

The convertfluxpol function is used to convert the flux to another polarization representation. There are are three representations used, namely , 'Stokes', 'linear' & 'circular'

Arguments

Inputs		
which	A vector of indices specifying the components to modify. A vector with indices between 0 and one less than the list length, inclusively	
	allowed:	int
	Default:	
polarization	The new polarization representation	
	allowed:	string
	Default:	circular
		linear
		Stokes

Returns
bool

Example

```
cl.open('centarusA.cl')
print cl.getfluxvalue(1)
cl.convertfluxpol(1, 'linear')
print cl.getfluxvalue(1)
```

componentlist.getrefdir.html

componentlist.getrefdir - Function

1.2.1 Return the reference direction

Description

The getrefdir function returns, as a direction measure, the reference direction for the specified component. The reference direction is for all the currently supported component shapes the direction of the centre of the component.

Arguments

Inputs	
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int Default:

Returns

record

Example

```
cl.open('crux.cl')
dir = cl.getrefdir(1)
```

---

componentlist.getrefdirra.html

componentlist.getrefdirra - Function

1.2.1 Get the RA of the reference direction. **(Not implemented not)**

Description

The getrefdirra function returns the right ascension of the reference direction of the component as a formatted string. If the reference frame is something other than J2000 or B1950 the value returned is the latitude or the azimuthal angle. The unit argument specifies the units for the returned value. It can be any angular unit (eg. 'deg', 'rad', 'arcsec', 'mas') or it can be 'angle' or 'time'. If it is 'angle' then the returned string is formatted in degrees, minutes, seconds ie., '+DDD.MM.SS.sss'. If it is 'angle' then the returned string is formatted in hours, minutes, seconds ie., 'HH:MM:SS.sss'.
The precision argument controls the numerical precision of the returned value. For the angular units it controls how many digits are in the returned string. For the 'angle' unit, precisions of two, four & six control whether the degrees, degrees,minutes or degrees, minutes & seconds are returned. Higher precisions increase the precision of the seconds field. Similarly, for the 'time' unit precisions of two, four & six control whether the hours, hours, minutes or hours, minutes & seconds are returned.
All directions are stored internally in double precision.

Arguments

Inputs	
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int Default:
unit	The angular unit of the returned value. Any string containing an angular unit or 'angle' or 'time' allowed: string Default: deg
precision	The number of digits in the returned string. Numbers between 1 and 16 make the most sense allowed: int Default: 6

Returns

string

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
print 'The first component is at RA: ', cl.getrefdirra(1, 'time'),
      ' Dec: ', cl.getrefdirdec(1, 'angle'),
      ' (', cl.getrefdirframe(1), ')'
```

---

componentlist.getrefdirdec.html

componentlist.getrefdirdec - Function

1.2.1 Get the declination of the reference direction.**(Not implemented yet)**

Description

The getrefdirdec function returns the declination of the reference direction of the component as a formatted string. If the reference frame is something other than J2000 or B1950 the value returned is the longitude or the altitude.
See the getrefdirra function for a description of the unit and precision arguments.

Arguments

Inputs	
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int Default:
unit	The angular unit of the returned value. Any string containing an angular unit or 'angle' or 'time' allowed: string Default: deg
precision	The number of digits in the returned string. Numbers between 1 and 16 make the most sense allowed: int Default: 6

Returns
string

Example

```
See the example for the
getrefdirra function.
```

componentlist.getrefdirframe.html

componentlist.getrefdirframe - Function

1.2.1 Get the reference frame of the reference direction.

Description

The getrefdirframe function returns the reference frame of the reference direction of the component as a string. The returned string is always in upper case. Common frames are, 'J2000', 'B1950' and 'GALACTIC'.

Arguments

Inputs	
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int Default:

Returns

string

Example

```
See the example for the
getrefdirra function.
```

---

componentlist.setrefdir.html

componentlist.setrefdir - Function

1.2.1 Set the reference direction

Description

The setrefdir function sets the reference direction of the specified components to a new value. The direction is defined by separately specifying the right ascension and the declination.
The right ascension is specified as a string or a real number
Ra can be in standard angle units 'deg', 'rad', or time formatted as such 'HH:MM:SS.sss' eg., '19:34:63.8' or angle formatted as such '+DDD.MM.SS.sss' eg., '127.23.12.37'.
Similarly the declination is specified as a string or a real number and the decunit can be any angular unit or 'angle' or 'time'.

Arguments

Inputs		
which	A vector of indices specifying the components to modify. A vector with indices between 0 and one less than the list length, inclusively	
	allowed:	int
	Default:	1
ra	The RA of the new direction, A formatted string or a number	
	allowed:	any
	Default:	
dec	The declination of the new direction. A formatted string or a number	
	allowed:	any
	Default:	
log	Send a message to the logger	
	allowed:	bool
	Default:	true

Returns

bool

Example

```
cl.simulate(3)
cl.setrefdir(0, '12:26:35.9', '-63.5.56')
cl.setrefdir(1, '12h26m35.9', '-63d5m56')
cl.setrefdir(2, '-173.35deg', '-1.10128rad')
cl.rename('testcls.cl')  # write to disk
```

---

componentlist.setrefdirframe.html

componentlist.setrefdirframe - Function

1.2.1 Set the reference frame for the direction

Description

The setrefdirframe function sets the reference frame for the reference direction of the specified components (what a mouthful)!
Currently the reference frame does not include additional information like when and where the observation took place. Hence only reference frames that are independent of this information can be used. This includes the common ones of 'J2000', 'B1950', and 'Galactic'. The measures module contains a complete listing of all possible reference frames. The parsing of the reference frame string is case-insensitive.

Arguments

Inputs	
which	A vector of indices specifying the components to modify. A vector with indices between 0 and one less than the list length, inclusively allowed: int Default:
frame	The new reference frame, A string like 'B1950', 'J2000' or 'galactic' allowed: string Default:
log	Send a message to the logger allowed: bool Default: true

Returns
bool

Example

```
cl.open('crux.cl');
cl.setrefdirframe(0, 'B1950');
```

---

componentlist.convertrefdir.html

componentlist.convertrefdir - Function

1.2.1 Convert the reference direction to a new frame

Description

The convertrefdir function changes the specified components to use a new direction reference frame. Using this function will change the right-ascension and declination of the component(s), unlike the setrefdirframe which does not. Please see the setrefdirframe function for a description of what frames are allowed.

Arguments

Inputs	
which	A vector of indices specifying the components to modify. A vector with indices between 0 and one less than the list length, inclusively allowed: int Default:
frame	The new reference frame A string like 'B1950', 'J2000' or 'galactic' allowed: string Default:

Returns

bool

Example

```
cl.open('crux.cl');
cl.convertrefdirframe(0, 'J2000');
```

componentlist.shapetype.html

componentlist.shapetype - Function

1.2.1 Returns the shape type of the component

Description

The shapetype function returns the current shape of the specified component. The shape defines how the flux of the component varies with direction on the sky. Currently there are three shapes available. These are 'Point', 'Gaussian', 'Disk', and 'Limbdarkeneddisk' (experimental). This function returns one of these four strings.

Arguments

Inputs	
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int Default:

Returns

string

Example

```
cl.open('crux.cl')
print 'The first component has a', cl.shapetype(0), ' shape'
```

componentlist.getshape.html

componentlist.getshape - Function

1.2.1 Return the shape parameters the component

Description

The getshape function returns the shape parameters of a component in a record. As different shapes can have a differing number and type of parameters the shape parameters are returned in a record with fields that correspond to parameters relevant to the current shape.
For a point shape there are only two fields; type and direction. These are the shape type, and the reference direction. These values are also returned by the shapetype and getrefdir functions.
For both the Gaussian and disk shapes there are three additional fields; majoraxis, minoraxis & positionangle. These are angular quantities, and hence are records with a value and a unit.

Arguments

Inputs	
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int Default:

Returns
record

Example

```
See the examples for the
setshape \&
convertshape
functions.
```

componentlist.setshape.html

componentlist.setshape - Function

1.2.1 Change the shape of the component

Description

The setshape function changes the shape of the specified components to the user specified shape.
The type argument defines what the sort of new shape to use. This can be either 'point', 'Gaussian', 'disk', or 'limbdarkeneddisk'. The parsing of this string is case insensitive. The 'limbdarkeneddisk' is an experimental disk model with the limb-darkening effect, where the sky brigtness is described as $I = I_o(1 - (r/R)^2)^{n/2}$ with R being apparent body radius. The n can be set in optionalparms (if it is not set, the default value, 0.0 will be used).
If the shape type is 'point' then the remaining arguments in this function are ignored. There are no other parameters needed to specify a point shape.
But if the shape is 'Gaussian', 'disk', or 'limbdarkeneddisk', the remaining arguments are needed to fully specify the shape. The majoraxis, minoraxis and positionangle arguments are quantities (see the quanta module for a definition of a quantity). Hence they can be specified either as with string eg., '1arcsec' or with a record eg., [value=1, unit='deg'].
The major axis is the width of the larger axis. For the Gaussian shape this is the full width at half maximum. And the minor axis is the width of the orthogonal axis. The positionangle is the specifies the rotation of these two axes with respect to a line connecting the poles of the current direction reference frame. If the angle is positive the the north point of the component moves in the eastern direction.

Arguments

Inputs	
which	A vector of indices specifying the components to modify. A vector with indices between 0 and one less than the list length, inclusively allowed: int Default:
type	The new shape type. A string that is either 'point', 'Gaussian', 'disk'i, or 'limbdarkeneddisk' allowed: string Default: disk limbdarkeneddisk Gaussian Point
majoraxis	The width of the larger axis. A quantity with angular units allowed: any Default: 1.0arcmin
minoraxis	The width of the smaller axis. A quantity with angular units allowed: any Default: 1.0arcmin
positionangle	The rotation of the axes with respect to the reference frame. A quantity with angular units allowed: any Default: 0.0deg
majoraxiserror	Error ∼The width of the larger axis. A quantity with angular units allowed: any Default: 0.0arcmin
minoraxiserror	Error of the width of the smaller axis. A quantity with angular units allowed: any Default: 0.0arcmin
positionangleerror	Error of the rotation of the axes with respect to the reference frame. A quantity with angular units allowed: any Default: 0.0deg
optionalparms	optional parameters in a vector (for limbdarkeneddisk) allowed: doubleArray Default: 0.0
log	Send a message to the logger allowed: bool Default: true

Returns

bool

Example

```
cl.open('crux.cl', nomodify=False)
cl.setshape(3, 'disk',  '45mas', '45mas')
print cl.getshape(3)['majoraxis']
```

---

componentlist.convertshape.html

componentlist.convertshape - Function

1.2.1 Change the units of the shape parameters **(Not implemented yet)**

Description

The convertshape function changes the units of the specified shape parameters on the specified components. When changing the units it also converts the values so that overall the angle has not changed.
Depending on the component shape some arguments of this function are ignored. If the shape type is 'point', then all but the which argument are ignored. This function is useless for points.
If the shape is a 'gaussian' or 'disk' then this will modify the units of the major and minor axes and the positionangle. Use the getshape function to see these parameters using the new units.

Arguments

Inputs		
which	A vector of indices specifying the components to modify. A vector with indices between 0 and one less than the list length, inclusively	
	allowed:	int
	Default:	
majoraxis	The units to use on the larger axis. A string with angular units	
	allowed:	string
	Default:	rad
		deg
		mas
		arcsec
		arcmin
minoraxis	The units to use on the smaller axis. A string with angular units	
	allowed:	string
	Default:	rad
		deg
		mas
		arcsec
		arcmin
positionangle	The units to use for the rotation of these axes. A string with angular units	
	allowed:	string
	Default:	rad
		deg

Returns
bool

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
cl.convertshape(3, 'arcsec', 'arcsec');
print cl.getshape(3).minoraxis;
```

componentlist.spectrumtype.html

componentlist.spectrumtype - Function

1.2.1 Returns the spectral shape of the component

Description

The spectrumtype function returns the current spectral shape of the specified component. The spectral shape defines how the flux of the component varies with frequency. Currently there are two spectral shapes available. These are 'Constant' and 'Spectral Index'. This function returns one of these two strings.

Arguments

Inputs	
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int Default:

Returns

string

Example

```
cl.open('crux.cl')
print 'The first component has a', cl.spectrumtype(1), ' spectrum'
```

componentlist.getspectrum.html

componentlist.getspectrum - Function

1.2.1 Return the spectral parameters the component

Description

The getspectrum function returns the spectral parameters of a component in a record. As different spectral shapes can have a differing number and type of parameters the spectral parameters are returned in a record with fields that correspond to parameters relevant to the current spectral shape.
For a constant spectrum there are only two fields; type and frequency. These are the spectral type, and the reference frequency. These values are also returned by the spectrumtype and getfreq functions.
For the spectral index spectral shape there is also an index field. This contains a vector with four numbers, these are the spectral indicies for the I,∼Q,∼U,∼&∼V components of the flux.

Arguments

Inputs	
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int Default:

Returns

record

Example

```
See the examples for the
setspectrum \&
getspectrum
functions.
```

componentlist.setstokesspectrum.html

componentlist.setstokesspectrum - Function

1.2.1 Change the spectrum of the component

Description

The setstokesspectrum function changes the spectrum of the specified components to the user specified spectrum. This is different from setspectrum as it provides ways to control variation of all 4 Stokes parameters with frequency. If only I variation is needed please use setspectrum
The type argument defines what the sort of new spectrum to use. This can be either 'constant' or 'spectral index' or 'tabular'. The parsing of this string is case insensitive.
If the spectrum type is 'constant' then the remaining arguments in this function are ignored. There are no other parameters needed to specify a constant spectrum.
But if the spectrum is 'spectral index', the `index` argument is needed. It is a 4 element vector.
The first element (α_0) is the spectral index of stokes I ($I(\nu) = I(\nu_0)(\frac{\nu}{\nu_0})^{\alpha_0}$)
The second element (α_1) is a spectral index for the fractional linear polarization ($\sqrt{\frac{(Q(\nu)^2+U(\nu)^2)}{I(\nu)^2}} = \sqrt{\frac{(Q(\nu_0)^2+U(\nu_0)^2)}{I(\nu_0)^2}}(\frac{\nu}{\nu_0})^{\alpha_1}$). $\alpha_1 = 0$ implies constant fractional linear polarization w.r.t frequency.
The third element is a "Rotation Measure" factor, i.e angle of rotation $\theta = \alpha_2(\lambda^2 - \lambda_0^2)$ of the linear polarization at frequency ν w.r.t frequency ν_0.
The fourth element is a spectral index for the fractional spectral polarization ($\frac{V(\nu)}{I(\nu)} = \frac{V(\nu_0)}{I(\nu_0)}(\frac{\nu}{\nu_0})^{\alpha_3}$
If the spectrum is 'tabular', then `index` is ignored but the six parameters `tabularfreqs, tabulari, tabularq, tabularu, tabularv and tabularframe` are considered. `tabularfreqs` and `tabulari, tabularq, tabularu, tabularv` have to be list of same lengths and larger than 2. You need at least 2 samples to interpolate the spectral value in between. The Stokes parameters of the source is interpolated from these values. Extrappolation outside the range given in `tabularfreqs` is not done. `tabularfreqs` should be float values in 'Hz' `tabulari, tabularq, tabularu, tabularv` should be float values in 'Jy'
You should ensure that the reference frequency is set to the value you desire, using the setfreq function if you change to the spectral index shape.

Arguments

Inputs		
which	The index specifying the component to modify. A value between 0 and one less than the list length, inclusively	
	allowed:	int
	Default:	
type	The new spectrum type. A string that is either 'constant or 'spectral index' or 'tabular'	
	allowed:	string
	Default:	spectral index
index	The spectral index.	
	allowed:	doubleArray
	Default:	0.0
tabularfreqs	The frequencies of for the tabular values, in Hz	
	allowed:	doubleArray
	Default:	1.0e11
tabulari	tabular Stokes I values, in Jy (same length as tabularfreqs)	
	allowed:	doubleArray
	Default:	1.0
tabularq	tabular Stokes Q values, in Jy (same length as tabularfreqs)	
	allowed:	doubleArray
	Default:	0.0
tabularu	tabular Stokes U values, in Jy (same length as tabularfreqs)	
	allowed:	doubleArray
	Default:	0.0
tabularv	tabular Stokes V values, in Jy (same length as tabularfreqs)	
	allowed:	doubleArray
	Default:	0.0
reffreq	The reference frequency for spectral index	
	allowed:	any
	Default:	1.4GHz
frame	The frame for which the frequencies given are in	
	allowed:	string
	Default:	LSRK

Returns

bool

Example

```
  This example add a point source model and revises the model point source spectral vari
 I is assigned a spectral index of 1. fractional linear pol is assigned a spectral index

cl.addcomponent(shape='point', flux=[10.0, 0.4, -0.2, 0.1], dir='J2000 19h00m00 -20d00m0
cl.setstokesspectrum(which=0, type='spectral index', index=[1.0, 0.4, 0, 0.4], reffreq='
cl.rename('my19hcomp.cl')
cl.done()
```

```
In this example a componentlist  is created from scratch and 2 sources are added
One whose spectral variation is defined by a spectral index and the other one as
tabular values. Both components have full Stokes parameters spectral variation
defined.

cl.done()  ### effectively resets state of cl tool
###add first component
cl.addcomponent(flux=[10, 0.5, -0.3, 0.2], dir='J2000 15h22m00 5d04m00')
cl.setstokesspectrum(which=0, type='spectral index', index=[1.0, 0.4, 0, 0.4], reffreq='2
###adding second component; flux value is unimportant as the tabular values will
###will set it
cl.addcomponent(flux=[1.0, 0, 0, 0],dir='J2000 15h22m30 5d05m00')
##define the IQUV flux variation as tabular values at different frequencies.
cl.setstokesspectrum(which=1, type='tabular', tabularfreqs=[1.0e9, 1.1e9, 1.2e9, 1.3e9, 1
###saving the componentlist to disk
cl.rename('two_comp.cl')
cl.done()  ###done is needed to sync to disk
```

componentlist.setspectrum.html

componentlist.setspectrum - Function

1.2.1 Change the spectrum of the component

Description

The setspectrum function changes the spectrum of the specified components to the user specified spectrum.
The type argument defines what the sort of new spectrum to use. This can be either 'constant'or 'spectral index'. The parsing of this string is case insensitive.
If the spectrum type is 'constant' then the remaining arguments in this function are ignored. There are no other parameters needed to specify a constant spectrum.
But if the spectrum is 'spectral index', the `index` argument is needed to fully specify the spectrum by using the index argument.
If the spectrum is 'tabular', then `index` is ignored but the three parameters `tabularfreqs, tabularflux and tabularframe` are considered.
`tabularfreqs` and `tabularflux` have to be list of same lengths and larger than 2. You need at least 2 samples to interpolate the spectral value in between. The flux of the source is interpolated from these values.
Extrappolation outside the range given in `tabularfreqs` is not done.
`tabularfreqs` should be float values in 'Hz' `tabularflux` should be float values in 'Jy'
You should ensure that the reference frequency is set to the value you desire, using the setfreq function if you change to the spectral index shape.

Arguments

Inputs		
which	The index specifying the component to modify. A value between 0 and one less than the list length, inclusively	
	allowed:	int
	Default:	
type	The new spectrum type. A string that is either 'constant or 'spectral index' or 'tabular'	
	allowed:	string
	Default:	spectral index
index	The spectral index.	
	allowed:	double
	Default:	0.0
tabularfreqs	The frequencies of for the tabular values, in Hz	
	allowed:	doubleArray
	Default:	1.0e11
tabularflux	tabular flux density values, in Jy (same length as tabularfreqs)	
	allowed:	doubleArray
	Default:	1.0
tabularframe	The frame for which the frequencies given are in	
	allowed:	string
	Default:	LSRK

Returns
bool

Example

```
cl.open('centarusA.cl')
cl.setspectrum(2, 'spectral index', -0.5)
print cl.getcomponent(2)['spectrum']['index']
cl.done()
```

This example revises the model for Centaurus-A changing the spectral index of all the components in the left lobe. The output from the print statement is \verb|[-0.5 0 0 0]|

```
cl.addcomponent(shape='point', flux=[1.0, 0.0, 0.0, 0.0], dir='J2000 19h00m00 -20d00m00
cl.setspectrum(which=0, type='tabular', tabularfreqs=[1.0e9, 1.1e9, 1.4e9], tabularflux=
cl.rename('my19hcomp.cl')
cl.done()
```

In this example a component is created from scratch as a point source
The spectrum is set to, say, measured values at 3 frequencies (1GHz, 1.1GHz and 1.4GHz) t
Any frequency in between the range 1 to 1.4 GHz the flux will be estimated by interpolat

componentlist.getfreq.html

componentlist.getfreq - Function

1.2.1 Get the reference frequency **(Not implemented yet)**

Description

The getfreq function returns, as a frequency measure, the reference frequency for the specified component. At the reference frequency the flux of the component is the value obtained with the getfluxvalue function. The flux may be different at other frequencies, depending on the components spectral shape. If the spectral shape is constant then changing the reference frequency will not affect the spectrum of the component.

Arguments

Inputs	
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int Default:

Returns

record

Example

```
include 'componentlist.g'
cl := componentlist('centarusA.cl');
f := cl.getfreq(2);
```

componentlist.getfreqvalue.html

componentlist.getfreqvalue - Function

1.2.1 Get the reference frequency value **(Not implemeted yet)**

Description

The getfreqvalue function returns as a floating point number the value of the reference frequency. To fully interpret this value you should also use the frequency unit, obtained using the getfrequnit function and the frequency reference frame, obtained using the getfrequnit function

Arguments

Inputs	
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int Default:

Returns

double

Example

```
include 'componentlist.g'
cl := componentlist('centarusA.cl');
print 'The reference frequency is ', cl.getfreqvalue(1), ' ',
      cl.getfrequnit(1), '(', cl.getfreqframe(1), ')'
```

componentlist.getfrequnit.html

componentlist.getfrequnit - Function

1.2.1 Get the reference frequency unit **(Not implemeted yet)**

Description

The getfrequnit function returns as a string that defines the units of the reference frequency. This unit should be used in conjunction with the getfreqvalue function.

Arguments

Inputs	
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int Default:

Returns

string

Example

```
See the example for the
getfreqvalue function.
```

---

componentlist.getfreqframe.html

componentlist.getfreqframe - Function

1.2.1 Get the reference frequency frame **(Not implemeted yet)**

Description

The getfreqframe function returns as a string the reference frame of the reference frequency of the specified component.
See the measures module for a description of the available frequency reference frames. Common frames are, 'LSR', 'TOPO' and 'GEO'.
The frame string is always returned in upper case.

Arguments

Inputs	
which	An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int Default:

Returns
string

Example

```
See the example for the
getfreqvalue function.
```

---

componentlist.setfreq.html

componentlist.setfreq - Function

1.2.1 Set the reference frequency

Description

The setfreq function sets the reference frequency of the specified components to a new value. The frequency is defined by separately specifying the value and its units. Use the setfreqframe function to set the frequency reference frame

Arguments

Inputs		
which	An index specifying the component to modify An integer between 0 and one less than the list length, inclusively	
	allowed:	int
	Default:	no default
value	The new frequency value. A number	
	allowed:	double
	Default:	
unit	The units of the frequency. Any string with the same dimensions as the 'Hz'	
	allowed:	string
	Default:	'GHz'
log	Send a message to the logger	
	allowed:	bool
	Default:	true

Returns

bool

Example

```
cl.open('centarusA.cl')
cl.setfreq(1, 1.415, 'GHz')
```

componentlist.setfreqframe.html

componentlist.setfreqframe - Function

1.2.1 Set the reference frame for the frequency

Description

The setfreqframe function sets the reference frame for the reference frequency of the specified components.
Currently the reference frame does not include additional information like when are where the observation took place. Hence no conversion between reference frames is available. In the interim I recommend you always use the same frame.

Arguments

Inputs		
which	An index specifying the component to modify. An integer between 0 and one less than the list length, inclusively	
	allowed:	int
	Default:	
frame	The new reference frame, A string like 'LSRK', 'LSRD', 'GEO' or 'TOPO	
	allowed:	string
	Default:	TOPO
		GEO
		LSRD
		LSRK
log	Send a message to the logger	
	allowed:	bool
	Default:	true

Returns
bool

Example

```
cl.open('centarusA.cl')
cl.setfreqframe(0, 'LSRK')
```

---

componentlist.convertfrequnit.html

componentlist.convertfrequnit - Function

1.2.1 Convert the reference frequency to a new unit

Description

The convertfrequnit function changes the specified components to use a new unit for the reference frequency. Using this function will change the frequency value also so that the overall reference frequency is not changed. It will affect the values and units obtained with setfreqvalue function.
Any unit can be used that has the same dimensions as the 'Hz'.

Arguments

Inputs	
which	An index specifying the component to modify. An integer between 0 and one less than the list length, inclusively allowed: int Default:
unit	The new frequency unit. Any string with the same dimensions as the 'Hz' allowed: string Default: 'GHz'

Returns
bool

Example

```
cl.open('centarusA.cl');
cl.convertfrequnit(1, 'kHz');
```

componentlist.getcomponent.html

componentlist.getcomponent - Function

1.2.1 Extract a component from the list.

Description

The component function returns a record, showing the current state of the specified component in the list.
Modifying the record that is returned by this function does not modify the component in the list. To do this you must remove the component from the list, using the remove function, and add the modified component using the add function, or use the replace object function. This function will be removed in a future release of aips++ and you are urged to use the get functions to extract information about a component.

Arguments

Inputs		
which	index of which component to extract. integers between 0 and one less than the length of the list, inclusively	
	allowed:	int
	Default:	
iknow	Suppress the warning message	
	allowed:	bool
	Default:	false

Returns

record

---

componentlist.add.html

componentlist.add - Function

1.2.1 Add a component to the list.

Description

The add function adds a component to the component list. You cannot add components to a list that has been opened read only. To use this function you need to know the details of the record format. however this has been deprecated and you are urged to use the set functions, in conjunction with the simulate function to add a component to the list.

Arguments

Inputs	
thecomponent	A record that represents a component. any record that contains the required fields allowed: record Default:
iknow	Suppress the warning message allowed: bool Default: true

Returns

bool

---

componentlist.replace.html

componentlist.replace - Function

1.2.1 Replace components in the list. **(Not implemented yet)**

Description

The replace function replaces the components from the list with the specified components from another list. The source list can be opened readonly and the length of the vectors in the first and third arguments must the the name.
You cannot replace components in a list that has been opened read only.

Arguments

Inputs	
which	A vector of indices specifying the components to replace. A vector with indices between 0 and one less than the list length, inclusively allowed: int Default:
list	The list containing the components to copy. A componentlist tool allowed: record Default:
whichones	A vector of indices specifying the components to copy A vector with indices between 1 and the length of the list in the second argument allowed: intArray Default: -1

Returns

bool

---

componentlist.summarize.html

componentlist.summarize - Function

1.2.1 Summarize the specified component to the logger

Description

The summarize function summarizes the contents of the specified components to the logger.

Arguments

Inputs	
which	An index specifying which component. Unset or an integer between 0 and one less than the list length, inclusive allowed: int Default: -1

Returns

bool

---

componentlist.iscomponentlist.html

componentlist.iscomponentlist - Function

1.2.1 Is the argument a componentlist tool? **(Not implemented yet)**

Description

This global function can be used to determine if the supplied argument is a componentlist tool. If so it returns True, otherwise it returns False.

Arguments

Inputs		
tool	The variable that you wish to test	
	allowed:	any
	Default:	

Returns

bool

Example

```
include 'componentlist.g'
if (iscomponentlist(cl)) {
  cl.simulate(2);
} else {
  fail 'Not a componentlist';
}
```

---

ms-Module.html

1.3 ms - Module

Module for measurement set operations

Description A CASA measurement set is a CASA table which obeys specific conventions. These conventions are defined in note 229. Like all CASA tables the measurement set will always appear as a directory which contains a number of files and directories.

Measurement set tables come in two slightly different versions, single dish and interferometric. Single dish measurement sets store the observed data as real numbers in the `FLOAT_DATA` column of the measurement set, whereas interferometric ones use complex numbers in the `DATA` column.

A measurement set table can contain data from a variety of different observations with different spectral or polarimetric configurations, different pointings and different instruments. To do this it needs to handle data with differing shapes. The data shape referred to here is two-dimensional with the length of the axes being the number of correlations and the number of channels in the data. A typical shape might be `[4, 1]`, which could correspond to a continuum observation where the `[RR, LL, RL, LR]` polarizations where correlated. In the same measurement set there may be data with a shape of `[1, 32]`, which corresponds to a spectral line observation, with 32 channels, where only the `[XX]` polarizations are correlated.

ms-Tool.html

1.3.1 ms - Tool

Operations on measurement sets
Requires: table **Synopsis**

Description

The ms tool provides functions to manipulate the contents of measurement set tables. The functions can be categorised as shown below.

Attaching to a Measurement Set
The simplest and most common way to attach an ms tool to a measurement set is to use the ms.open function which requires that you specify the name of the measurement set table.
The function ms.fromfits converts a UVFITS file to a measurement set table prior to attaching the ms tool to the newly created measurement set. The conversion step may take some time if the FITS file is large. However it only needs to be done once. The measurement set table is not deleted when you close the ms tool, using the close function, or exit CASA. And once the measurement set table is created it is much faster to attach an ms tool to it using the ms.open function.

Getting summary information
The summary function will display, in the logger, an overview of the measurement set. This will include listings of the fields, spectral windows & polarization setups used in the measurement set.
The range function will provide more quantitative information on the minimum, maximum or used values of specified parameters. When using this function you may need to do an initial selection, as described below, depending on whether the parameters you ask for change their shape. A list of parameters accepted by the range function is given in table∼1.6 and this table also indicates when an initial selection is necessary.
The lister function provides a concise listing of the data in the measurement set.
The listhistory function lists the contents of the measurement set history table. The history table contains a record of changes made to the measurement set by flagger, calibrater, imager and other tools.

Selecting data
As described in the ms module documentation a measurement set can contain data with a variety of different shapes. Some of the functions in this tool require the data to be in a fixed shape. Before you can use these functions you

need to select a subset of the data in the measurement set where all the data has a fixed shape. There are two functions which can be used to do this. These are the selectinit and command functions.
The select function can be used to further refine which subset of the data will be used by the data access functions. This function allows you to select specific rows in a measurement set using a wide range of criteria.
The select function can only select whole rows in a measurement set. To select specific channels within a row you use the selectchannel function. Similarly to select specific polarizations you should use the selectpolarization function.

Reading and writing data
The getdata function is used to read data from the measurement set into casapy variables. You can select which columns of the measurement set main table you are interested in and only the subset of data specified using the selection functions described above will be retreived. Any frequency averaging (see the selectfrequency function) and polarization conversion (see the selectpolarization function) will be done when you retreive the data. The full power of casapy and other CASA tasks and tools, like plotxy, can then be used for adhoc inspection and calculations involving the data.
If the measurement set was opened for writing then the putdata function can be used to write the data back into the measurement set. When writing data back into the measurement set you cannot change the data shape or the coordinates of the data, only the numerical values. This means that you cannot write data that has been averaged in frequency or converted to different polarizations.
When using the getdata function with a large measurement set you need to be careful to not request too much data. The measurement set is stored on disk but casapy variables are stored in memory. To allow you to access large amounts of data in an ordered way the ms tool provides functions that allow you to iterate through the data in a convienient way.
If you need to step through the data in an orderly fashion, you can use the iteration functions. These allow you to set up an iteration order (iterinit), obtain the first iteration (iterorigin), go to the next iteration (iternext) and end the iteration prematurely (iterend). The iterorigin and iternext function set the currently selected table (as used by getdata and others) to the current iteration. At the end of the iteration, the original selection is restored.
You can iterate through a measurement set you have previously selected using select, but if you use select while iterating, you cannot get back the unselected iteration (without reiterating through the table until the current point).
The writehistory function allows messages to be appended to the measurement set history table should you wish to do so. The listhistory function lists your messages and those created by flagger, calibrater, imager and other tools.

Conversions to FITS
Just as the fromfits function will convert a UVFITS file to a measurement set the tofits function will convert a measurement set to a UVFITS file. Similarly a single dish measurement set ie., one with a FLOAT_DATA column rather

than a DATA column, can be converted to a single dish FITS file using the tosdfits function.
You cannot read a UVFITS file into a measurement set and write it out as a single dish FITS file or vice-versa.

Concatenation
The concatenate function can be used to append the data from one measurement set to the end of another. As all the data is copied this function may take some time if the measurement set to be copied is large. The measurement set needs to be opened for writing for this to work.
The virtconcatenate function enables virtual concatenation, i.e. the data is not rewritten but just reindexed such that the two input MSs have the same subtables. They can then be turned into a multi-MS.

Sorting the main table by time
The timesort function permits you to sort the MS main table by time in ascending order. This can be useful after a concatenation.

Sorting the main table by a custom set of columns
The sort function permits you to sort the MS main table by a custom set of columns in ascending order. This can be useful to compare tables generated in different ways (e.g.: cvel and mastransform)

Splitting The split function allows you to make a new ms from a subset of the actual ms.

Flagging data
The flag and buffer functions all belong together. The idea is to fill a buffer of data (fillbuffer), optionally retrieve it as a record (getbuffer) and display it in some data display tool, do operations like differencing (diffbuffer), clipping (clipbuffer) and manual edits of the displayed data (setbufferflags), and then write the flags back to the MS (writebufferflags), into the appropriate flag level, so you can choose to apply or undo them. The data is untouched by these functions.

Table 1.6: Items recognized by the range, select, getdata and putdata functions.
Items marked with a † are only available in interferometric measurement sets.
Items marked with a ‡ are only available in interferometric measurement sets
that have been processed with calibrator or imager. Items marked with a ∗ do
not require all the data in the selected measurement set to have the same shape.

& range	select	getdata	putdata	comment	
amplitude†	+	-	+	-	amplitude of observed data
corrected_amplitude‡	+	-	+	-	amplitude of corrected data

Continued from previous page					
	range	select	getdata	putdata	comment
model_amplitude‡	+	-	+	-	amplitude of model data
ratio_amplitude‡	-	-	+	-	amplitude of corrected/model
residual_amplitude‡	-	-	+	-	amplitude of residual data
obs_residual_amplitude‡	-	-	+	-	amplitude of obs residual data
antenna1	*	+	+	-	1st antenna id
antenna2	*	+	+	-	2nd antenna id
antennas	*	-	-	-	list of antenna names
array_id	*	+	-	-	
axis_info	-	-	+	-	description of data axes
chan_freq	+	-	-	-	channel frequencies
corr_names	+	-	-	-	list of polarization strings
corr_types	+	-	-	-	list of polarization enum values
data†	-	-	+	+	complex observed data
corrected_data‡	-	-	+	+	complex corrected data
model_data‡	-	-	+	+	complex model data
ratio_data‡	-	-	+	-	complex corrected/model
residual_data‡	-	-	+	+	complex residual data
obs_residual_data‡	-	-	+	+	complex observed residual data
feed1	*	+	+	-	1st feed in correlation
feed2	*	+	+	-	2nd feed in correlation
field_id	*	+	+	-	field number
fields	*	-	-	-	list of field names
flag	-	-	+	+	data flags
flag_row	-	-	+	+	MS row flags
flag_sum	-	-	+	-	flag summary
ha	-	-	+	-	add hour angle to axis_info
ifr_number	*	+	+	-	1000*antenna1+antenna2
imaginary†	+	-	+	-	imag part of observed data
corrected_imaginary‡	+	-	+	-	imag part of corrected data
model_imaginary‡	+	-	+	-	imag part of model data
ratio_imaginary‡	-	-	+	-	imag part of corrected/model
residual_imaginary‡	-	-	+	-	imag part of residual data
obs_residual_imaginary‡	-	-	+	-	imag part of obs residual data
imaging_weight‡	+	-	+	+	weights used for imaging
last	-	-	+	-	add LAST to axis_info
num_corr	+	-	-	-	number of polarizations
num_chan	+	-	-	-	number of freq channels
phase†	+	-	+	-	phase of observed data
corrected_phase‡	+	-	+	-	phase of corrected data
model_phase‡	+	-	+	-	phase of model data
ratio_phase‡	-	-	+	-	phase of corrected/model
residual_phase‡	-	-	+	-	phase of residual data
obs_residual_phase‡	-	-	+	-	phase of observed residual data
phase_dir	+	-	-	-	list of phase centers & epoch

Continued from previous page					
	range	select	getdata	putdata	comment
real†	+	-	+	-	real part of observed data
corrected_real‡	+	-	+	-	real part of corrected data
model_real‡	+	-	+	-	real part of model data
ratio_real‡	-	-	+	-	real part of corrected/model
residual_real‡	-	-	+	-	real part of residual data
obs_residual_real‡	-	-	+	-	real part of observed res. data
ref_frequency	+	-	-	-	reference frequency
rows	*	+	-	-	row numbers in *original* table
scan_number	*	+	+	-	
sigma	*	-	+	+	sigma of the data
data_desc_id	*	+	+	-	
time	*	+	+	-	MJD time range in seconds
times	*	+	-	-	list of MJD timeslots
ut	-	-	+	-	add UT to axis_info
uvw	-	-	+	-	uvw vector
u	*	+	+	-	u coordinate
v	*	+	+	-	v coordinate
w	*	+	+	-	w coordinate
uvdist	*	+	+	-	uv distance
weight	*	-	+	+	weight of the data

Example:

```
ms.open("3C273XC1.MS",nomodify=False)
ms.selectinit(datadescid=0)
ms.select({'antenna1':[1,3,5],'uvdist':[1200.,1900.]})
rec=ms.getdata(["weight","data"])
# modify rec['weight'] and rec['data'] values as desired
ms.putdata(rec)
ms.close()
```

We open the MS for writing, select an array and spectral window and then select a few antennas and a uv range. We then get out the weight values and the data. We change these values in casapy and then write them back to the measurement set. Finally, we close the ms, causing the values to be written back to disk.

Methods

fromfits	Create a measurement set from a uvfits file
fromfitsidi	Create a measurement set from a fits-idi file
listfits	
createmultims	

ismultims	
getreferencedtables	
getfielddirmeas	Returns the direction measure from the given FIELD table column and row
nrow	Returns the number of rows in the measurement set
iswritable	Returns True is the underlying Table is writable
open	Attach the ms tool to a measurement set table
reset	Re-attach the tool to the original MS.
close	Detach the ms tool from the measurement set table
name	Name of the measurement set table the tool is attached to.
tofits	Convert a measurement set to a uvfits file
summary	(PARTIALLY IMPLEMENTED!!!) Summarize the measurement set
getscansummary	Get the summary of the ms
getspectralwindowinfo	Get a summary of the spectral windows
listhistory	List history of the measurement set
writehistory	Add a row of arbitrary information to the measurement set history table
statistics	Get statistics on the selected measurement set
statistics2	Get statistics on the selected measurement set
range	Get the range of values in the measurement set
lister	List measurement set visibilities
metadata	Get the MS metadata associated with this MS.
selectinit	Initialize the selection of an ms
msselect	Use the MSSelection module for data selection.
msselectedindices	Return the selected indices of the MS database. The keys in the record are the sam
select	Select a subset of the measurement set.
selecttaql	Select a subset of the measurement set.
selectchannel	Select and average frequency channels
selectpolarization	Selection and convertion of polarizations
regridspw	transform spectral data to different reference frame and/or regrid the frequency cha
cvel	transform spectral data to different reference frame and/or regrid the frequency cha
cvelfreqs	calculate the transformed grid of the SPW obtained by combining a given set of SP
getdata	Read values from the measurement set.
putdata	Write new values into the measurement set
concatenate	Concatenate two measurement sets
testconcatenate	Concatenate only the subtables of two measurement sets excluding the POINTING
virtconcatenate	Concatenate two measurement sets virtually
timesort	Sort the main table of an MS by time
sort	Sort the main table of an MS using a custom set of columns
contsub	Subtract the continuum from the visibilities
statwt	Set WEIGHT and SIGMA from the scatter of the visibilities
split	make a new ms from a subset of an existing ms, adjusting subtables and indices
partition	make a new ms from a subset of an existing ms, without changing any subtables
iterinit	Initialize for iteration over an ms
iterorigin	Set the iterator to the start of the data.
iternext	Advance the iterator to the next lot of data
iterend	End the iteration and reset the selected table
fillbuffer	Fill the internal buffer with data and flags.

diffbuffer	Differentiate or difference the internal buffer.
getbuffer	Return the internal buffer as a Record for access from the intrepreter.
clipbuffer	(NON-FUNCTIONAL???) Clip the internal buffer with specified limits.
asdmref	Test if the MS was imported with option lazy=True in importasdm and optionally
setbufferflags	Set the flags in the buffer
writebufferflags	Write the flags in the internal buffer back to the table.
clearbuffer	Clear the internal buffer.
continuumsub	Continuum fitting and subtraction in uv plane
done	Closes the ms `tool`
msseltoindex	Returns ids of the selection used
hanningsmooth	Hanning smooth the frequency channels to remove Gibbs ringing.
uvsub	Subtract model from the corrected visibility data.
addephemeris	Connect an ephemeris table with the MS FIELD table
ngetdata	Read values from the measurement set. Use this method instead of the older getdat
niterinit	Initialize for iteration over an ms. Use this method instead of the older iterinit() m
niterorigin	Set the iterator to the start of the data. Use this method instead of the older iteror
niternext	Advance the iterator to the next lot of data. Use this method instead of the older i
niterend	Query if there are more iterations left in the iterator. Use this method instead of th

ms.fromfits.html

ms.fromfits - Function

1.3.1 Create a measurement set from a uvfits file

Description

This function will convert a UVFITS file to a measurement set table and then open the measurement set table. The newly created measurement set table will continue to exist after the tool has been closed.
Setting the lock argument to True will permanently lock the table preventing other processes from writing to the measurement set. Unless you expect this to happen, and want to prevent it, you should leave the lock argument at the default value which implies auto-locking.
Note that the variety of fits files that fromfits is able to interpret correctly is limited mostly to files similar to those produced by classic AIPS. In particular, it understands only binary table extensions for the antenna (AN), frequency (FQ) and source (SU) information and ignores other extensions.
This function returns True if it successfully attachs the ms tool to a newly created Measurement Set or False if something went wrong, like an error in a file name.

Arguments

Inputs	
msfile	Filename for the newly created measurement set
	allowed: string
	Default:
fitsfile	uvfits file to read
	allowed: string
	Default:
nomodify	open for read access only
	allowed: bool
	Default: true
lock	lock the table for exclusive use
	allowed: bool
	Default: false
obstype	specify the observation type: 0=standard, 1=fastmosaic, requiring small tiles in the measurement set
	allowed: int
	Default: 0
host	host to start ms tool on (IGNORED!!!)
	allowed: string
	Default:
forcenewserver	start a new server tool (IGNORED!!!)
	allowed: bool
	Default: false
antnamescheme	For VLA only, antenna name scheme, old style is just antenna number, new style prepends VA or EV
	allowed: string
	Default: old

Returns
bool

Example

```
ms.fromfits("3C273XC1.MS", "3C273XC1.fits")
```

---

ms.fromfitsidi.html

ms.fromfitsidi - Function

1.3.1 Create a measurement set from a fits-idi file

Description

This function will convert a UVFITS file to a measurement set table and then open the measurement set table. The newly created measurement set table will continue to exist after the tool has been closed.
Setting the lock argument to True will permanently lock the table preventing other processes from writing to the measurement set. Unless you expect this to happen, and want to prevent it, you should leave the lock argument at the default value which implies auto-locking.
Note that the variety of fits files that fromfits is able to interpret correctly is limited mostly to files similar to those produced by classic AIPS. In particular, it understands only binary table extensions for the antenna (AN), frequency (FQ) and source (SU) information and ignores other extensions.
This function returns True if it successfully attachs the ms tool to a newly created Measurement Set or False if something went wrong, like an error in a file name.

Arguments

Inputs		
msfile	Filename for the newly created measurement set	
	allowed:	string
	Default:	
fitsfile	fits-idi file to read	
	allowed:	string
	Default:	
nomodify	open for read access only	
	allowed:	bool
	Default:	true
lock	lock the table for exclusive use	
	allowed:	bool
	Default:	false
obstype	specify the observation type: 0=standard, 1=fastmosaic, requiring small tiles in the measurement set	
	allowed:	int
	Default:	0

Returns
bool

Example

```
ms.fromfits("3C273XC1.MS", "3C273XC1.fits")
```

---

ms.listfits.html

ms.listfits - Function

1.3.1

Description

Arguments

Inputs	
fitsfile	list HDU and typical data rows in a uvfits file
	allowed: string
	Default:

Returns

bool

Example

```
ms.listfits('ngc5921.fits')
```

---

ms.createmultims.html

ms.createmultims - Function

1.3.1

Description

Arguments

Inputs		
outputTableName		
	allowed:	string
	Default:	
tables		
	allowed:	stringArray
	Default:	
subtables		
	allowed:	stringArray
	Default:	
nomodify	prevent changes to the measurement set	
	allowed:	bool
	Default:	true
lock	lock the table for exclusive use by this tool	
	allowed:	bool
	Default:	false
copysubtables	copy the subtables from the first to all other member MSs	
	allowed:	bool
	Default:	false
omitsubtables	Omit the subtables from this list when copying subtables	
	allowed:	stringArray
	Default:	

Returns
bool

Example

ms.ismultims.html

ms.ismultims - Function

1.3.1

Description

Arguments

Inputs

Returns
bool

Example

---

ms.getreferencedtables.html

ms.getreferencedtables - Function

1.3.1

Description

Arguments

Inputs

Returns
stringArray

Example

---

ms.getfielddirmeas.html

ms.getfielddirmeas - Function

1.3.1 Returns the direction measure from the given FIELD table column and row

Description

This function returns the direction measures from the given direction column of the MS FIELD table as a either a measure dictionary or sexigesimal string representation.

Arguments

Inputs	
dircolname	Name of the direction column in the FIELD table allowed: string Default: PHASE_DIR
fieldid	Field ID, starting at 0 allowed: int Default: 0
time	(optional) time for ephemeris access (in seconds, as in Main table TIME column) allowed: double Default: 0
format	Output format. Either "measure" (measure dictionary) or "string" (sexigesimal representation). Minimum match supported. allowed: string Default: measure

Returns

anyvariant

Example

```
ms.open('3C273XC1.MS')
print "Delay direction from FIELD table row 3 =", ms.getfielddirmeas("DELAY_DIR", 3)
```

---

ms.nrow.html

ms.nrow - Function

1.3.1 Returns the number of rows in the measurement set

Description

This function returns the number of rows in the measurement set. If the optional argument selected is set to True, it returns the number of currently selected rows, otherwise it returns the the number of rows in the original measurement.

Arguments

Inputs		
selected	return number of selected rows	
	allowed:	bool
	Default:	false

Returns

int

Example

```
ms.open('3C273XC1.MS')
print "Number of rows in ms =", ms.nrow()
#Number of rows in ms = 7669
```

---

ms.iswritable.html

ms.iswritable - Function

1.3.1 Returns True is the underlying Table is writable

Description

This function returns True if the underlying MeasurementSet Table was opened for writing/update.

Arguments

Returns
bool

Example

```
ms.open('3C273XC1.MS',nomodify=False)
if ms.iswritable():
  print "MeasurementSet is writable"
else:
  print "MeasurementSet is readonly"
#MeasurementSet is writable
```

---

ms.open.html

ms.open - Function

1.3.1 Attach the ms tool to a measurement set table

Description

Use this function when you have detached (using the close function) the ms tool from a measurement set table and wish to reattach to another measurement set table.
If check=true, additional referential integrity checks on the MS are run. If any of these fail, an exception is thrown and the MS is not open (since it is not a valid MS).

Arguments

Inputs		
thems	Name of the measurement set table to open	
	allowed:	string
	Default:	
nomodify	prevent changes to the measurement set	
	allowed:	bool
	Default:	true
lock	lock the table for exclusive use by this tool	
	allowed:	bool
	Default:	false
check	Run additional internal integrity checks on the MS.	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
ms.open('3C273XC1.MS')
```

```
ms.close()
ms.open("anotherms", nomodify=False, lock=False)
```

---

ms.reset.html

ms.reset - Function

1.3.1 Re-attach the tool to the original MS.

Description

This function re-attaches the ms tool to the original MS, effectively discarding any prior operations, in particular any data selection operations using msselect function.

Arguments

Returns
bool

Example

---

ms.close.html

ms.close - Function

1.3.1 Detach the ms tool from the measurement set table

Description

This function detaches the ms tool from the associated measurement set table after flushing all the cached changes. After calling this function the ms tool is not associated with any measurement set and using any function other than open or fromfits will result in an error message being sent to the logger.
This function can be useful to avoid synchronization problems which can occur when different processes have the same ms open.

Arguments

Returns
bool

Example

```
See the example for the open function.
```

---

ms.name.html

ms.name - Function

1.3.1 Name of the measurement set table the tool is attached to.

Description

This function returns the name of the measurement set table that is being manipulated. If the ms tool is not attached to any measurement set then this function will return the string “none”.

Arguments

Returns
string

Example

```
ms.open('3C273XC1.MS')
print "Processing file", ms.name()
```

---

ms.tofits.html

ms.tofits - Function

1.3.1 Convert a measurement set to a uvfits file

Description

This function writes a UVFITS file that contains the data in the measurement set associated with this tool. The FITS file is always written in floating point format and the data are always stored in the primary array of the FITS file. If the measurement set has been imaged or calibrated in CASA, it may contain additional data columns. You need to select ONE of these columns to be written to the FITS file. The possible options are:

observed This is the raw data as collected by the telescope. All interferometric measurement sets must contain this column. A synonym for 'observed' is 'data'.

corrected This is the calibrated data. A synonym for 'corrected' is 'corrected_data'.

model This is the visibilites that would be measured using the current model of the sky. A synonym for 'model' is 'model_data'.

The parsing of these strings is case insensitive. If any other option is specified then the observed data will be written.
By default a single-source UVFITS file is written, but if the measurement set contains more than one field or if you set the multisource argument to True a multi-source UVFITS file will be written. Because of limitations in the UVFITS format you have to ensure that the data shape is fixed for all the data you intend to write to one FITS file. See the general description of this tool for how you can select data to meet this condition.
The combinespw argument is used to control whether data from different spectral windows will be written as different entries in the FITS FQ (frequency) table or combined as different IF's within one entry in the FQ table. You should normally only set this to True if you know that the data from different spectral windows were observed simultaneously, and the data in the measurement set can be equally divided between all the spectral windows (i.e. each window should have the same width). Use of this switch is recommended for data to be processed in classic AIPS and difmap (if possible, e.g., standard dual IF observations).
The padwithflags argument is only relevant if combinespw is True. If true, it will fill in data that is 'missing' with flags to fit the IF structure. This is

appropriate if the MS had a few frequency-dependent flags applied, and was then time-averaged by split. If the spectral windows were observed at different times, padwithflags=True will add a large number of flags, making the output file significantly longer. It does not yet support spectral windows with different widths.
The FITS GC (gain curve) and TY (system temperature) tables can be optionally written by setting the writesyscal argument to True. This is a rather WSRT-specific operation at the moment and may not work correctly for measurement sets from other telescopes.
The width argument is for channel averaging while outputing the data to the fits file. The default values of 1 will copy the channels of the input as is. The start channel number is the first channel of the spw expression. The number of output channels is determined by spw expression and the width. The width is the number of channels of the input data to make 1 channel of the output data.
One may overwrite the specified output file if it exists by specifying overwrite=True.

Arguments

Inputs	
fitsfile	Name of the new uvfits file allowed: string Default:
column	Data column to write, see above for options allowed: string Default: corrected
field	Field ids (0-based) or fieldnames to split out allowed: any Default: variant
spw	Spectral windows to split allowed: any Default: variant
width	number of input channels to average allowed: int Default: 1
baseline	Antenna names or Antenna indices to select allowed: any Default: variant
time	Limit data selected to be within a given time range. Syntax is the defined in the msselection link allowed: string Default:
scan	Limit data selected on scan numbers. Syntax is the defined in the msselection link allowed: any Default: variant
uvrange	Limit data selected on uv distance. Syntax is the defined in the msselection link allowed: any Default: variant
taql	For the TAQL experts, flexible data selection using the TAQL syntax allowed: string Default:
writesyscal	Write GC and TY tables allowed: bool Default: false
multisource	Write in multisource format allowed: bool Default: false
combinespw	Export spectral windows as IFs allowed: bool Default: false
writestation	Write station name instead of antenna name allowed: bool Default: false
padwithflags	If combinespw==True, pad data with flags to fit IFs allowed: bool Default: false
overwrite	Overwrite output file if it exists? allowed: bool Default: false

Returns
bool

Example

```
 ms.open('3C273XC1.MS')
 ms.tofits('3C273XC1.fits', column='DATA');
 ms.done()

This example writes  the observed data of a measurement set to a
\uvfits\ file.


 ms.open('big.ms')
 ms.tofits('part.fits', column='CORRECTED', field=[0,1], spw=[2]);
 ms.done()

This example writes part (the first 2 fields and the third spectral
window) of the corrected data to the fits file.
```

---

ms.summary.html

ms.summary - Function

1.3.1 (PARTIALLY IMPLEMENTED!!!) Summarize the measurement set

Description

This method will print a summary of the measurement set to the system logger. The verbose argument provides some control on how much information is displayed.
For especially large datasets, the cachesize parameter can be increased for possibly better performance.
This method can also return, in the header argument, a record containing the following fields.

nrow Number of rows in the measurement set

name Name of the measurement set

DESCRIPTION OF ALGORITHM TO CALCULATE THE NUMBER OF UNFLAGGED ROWS The number of unflagged rows will only be computed if listunflis True. The number of unflagged rows (the nUnflRows columns in the scans and fields portions of the listing) is calculated by summing the fractional unflagged bandwidth for each row (and hence why the number of unflagged rows, in general, is not an integer). Thus a row which has half of its total bandwidth flagged contributes 0.5 rows to the unflagged row count. A row with 20 of 32 channels of homogeneous width contributes $20/32 = 0.625$ rows to the unflagged row count. A row with a value of False in the FLAG_ROW column is not counted in the number of unflagged rows.

Arguments

Inputs		
verbose	Produce verbose logging output	
	allowed:	bool
	Default:	false
listfile	Output file	
	allowed:	string
	Default:	
listunfl	List unflagged row counts? If true, it can have significant negative performance impact.	
	allowed:	bool
	Default:	false
cachesize	EXPERIMENTAL. Maximum size in megabytes of cache in which data structures can be held.	
	allowed:	double
	Default:	50
overwrite	If True, tacitly overwrite listfile if it exists.	
	allowed:	bool
	Default:	false

Returns
record

Example

```
ms.open('3C273XC1.MS')
outr=ms.summary(verbose=True)
###print the begining of observation in this ms
print qa.time(qa.quantity(outr['header']['BeginTime'],'d'), form='ymd')
###print  a dictionary of the info of scan 1
outr['header']['scan_1']

This example will send a verbose summary of the measurement
set to the logger.
```

ms.getscansummary.html

ms.getscansummary - Function

1.3.1 Get the summary of the ms

Description

This function will return a summary of the main table as a structure

Arguments

Returns
record

Example

```
ms.open('3C273XC1.MS')
scanInfo = ms.getscansummary()
```

---

ms.getspectralwindowinfo.html

ms.getspectralwindowinfo - Function

1.3.1 Get a summary of the spectral windows

Description

This method will get a summary of the spectral window actually used in this ms. To be precise those reference by the data description table.

Arguments

Returns
record

Example

```
ms.open('3C273XC1.MS')
spwInfo = ms.getspectralwindowinfo()
```

---

ms.listhistory.html

ms.listhistory - Function

1.3.1 List history of the measurement set

Description

This function lists the contents of the measurement set history table.

Arguments

Returns
bool

Example

```
ms.open('3C273XC1.MS')
ms.listhistory()

The history table contents are listed in the logger.
```

---

ms.writehistory.html

ms.writehistory - Function

1.3.1 Add a row of arbitrary information to the measurement set history table

Description

This function adds a row to the history table of the specified measurement set containing any message that the user wishes to record. By default the history entry is written to the history table of the measurement set that is currently open, the message origin is recorded as 'MSHistoryHandler::addMessage()', the originating application is 'ms' and the input parameters field is empty.

Arguments

Inputs		
message	Message to be recorded in message field	
	allowed:	string
	Default:	
parms	String to be written to input parameter field	
	allowed:	string
	Default:	
origin	String to be written to origin field	
	allowed:	string
	Default:	MSHistoryHandler::addMessage()
msname	name of selected measurement set	
	allowed:	string
	Default:	
app	String to be written to application field	
	allowed:	string
	Default:	ms

Returns
bool

Example

```
  ms.open('3C273XC1.MS')
  ms.writehistory('an arbitrary history message')
  ms.listhistory()
```

A row is appended to the measurement set history table.

---

ms.statistics.html

ms.statistics - Function

1.3.1 Get statistics on the selected measurement set

Description

This function computes descriptive statistics on the measurement set. It returns the statistical values as a python dictionary. The given column name must be a numerical column. If it is a complex valued column, the parameter complex_value defines which derived real value is used for the statistics computation.

Arguments

Inputs		
column	Column name	
	allowed:	string
	Default:	
complex_value	Which derived value to use for complex columns (amp, amplitude, phase, imag, real, imaginary)	
	allowed:	string
	Default:	
useflags	Use the data flags	
	allowed:	bool
	Default:	true
spw	Spectral Window Indices or names : example : '1,2'	
	allowed:	string
	Default:	
field	Field indices or source names : example : '2,3C48'	
	allowed:	string
	Default:	
baseline	Baseline number(s): example: "2&3;4&5"	
	allowed:	string
	Default:	
uvrange	UV-distance range, with a unit : example : '2.0-3000.0 m'	
	allowed:	string
	Default:	
time	Time range, as MJDs or date strings : example : 'xx.x.x.x.x~yy.y.y.y.y'	
	allowed:	string
	Default:	
correlation	Correlations/polarizations : example : 'RR,LL,RL,LR,XX,YY,XY,YX'	
	allowed:	string
	Default:	
scan	Scan number : example : '1,2,3'	
	allowed:	string
	Default:	
array	Array Indices or names : example : 'VLAA'	
	allowed:	string
	Default:	
obs	Observation ID(s): examples : '' or '1~3'	
	allowed:	string
	Default:	

Returns
record

Example

```
ms.open("3C273XC1.MS")
ms.statistics(column="DATA", complex_value='amp', field="2")
```

---

ms.statistics2.html

ms.statistics2 - Function

1.3.1 Get statistics on the selected measurement set

Description

This function computes descriptive statistics on the measurement set. It returns the statistical values as a python dictionary. The given column name must be a numerical column. If it is a complex valued column, the parameter complex_value defines which derived real value is used for the statistics computation.

Arguments

Inputs		
column	Column name	
	allowed:	string
	Default:	
complex_value	Which derived value to use for complex columns (amp, amplitude, phase, imag, real, imaginary)	
	allowed:	string
	Default:	
useflags	Use the data flags	
	allowed:	bool
	Default:	true
useweights	Use the data weights	
	allowed:	bool
	Default:	false
spw	Spectral Window Indices or names : example : '1,2'	
	allowed:	string
	Default:	
field	Field indices or source names : example : '2,3C48'	
	allowed:	string
	Default:	
baseline	Baseline number(s): example: "2&3;4&5"	
	allowed:	string
	Default:	
uvrange	UV-distance range, with a unit : example : '2.0-3000.0 m'	
	allowed:	string
	Default:	
time	Time range, as MJDs or date strings : example : 'xx.x.x.x.x~yy.y.y.y.y'	
	allowed:	string
	Default:	
correlation	Correlations/polarizations : example : 'RR,LL,RL,LR,XX,YY,XY,YX'	
	allowed:	string
	Default:	
scan	Scan number : example : '1,2,3'	
	allowed:	string
	Default:	
intent	Scan intents : example : '*AMPL*,*PHASE*'	
	allowed:	string
	Default:	
array	Array Indices or names : example : 'VLAA'	
	allowed:	string
	Default:	
obs	Observation ID(s): examples : '' or '1~3'	
	allowed:	string
	Default:	
reportingaxes	Statistics reporting axes: example: 'ddid,field'	
	allowed:	string
	Default:	
timeaverage	Average data in time	
	allowed:	bool
	Default:	true
timebin	Time averaging interval	
	allowed:	string
	Default:	0s

Returns
record

Example

```
ms.open("3C273XC1.MS")
ms.statistics2(column="DATA", complex_value='amp', field="2")
```

---

ms.range.html

ms.range - Function

1.3.1 Get the range of values in the measurement set

Description

This function will return the range of values in the currently selected measurement set for the items specified. Possible items include most scalar columns, interferometer number (1000*antenna1+antenna2), uvdist(ance), u, v, w, amplitude, phase, real and imaginary components of the data (and corrected and model versions of these - if these columns are present). See the table at the top of the document to find the exact list.
You specify items in which you are interested using a string vector where each element is a case insensitive item name. This function will then return a record that has fields corresponding to each of the specified items. Each field will contain the range of the specified item. For many items the range will be the minimum and maximum values but for some it will be a list of unique values. Unrecognized items are ignored.
By default the FLAG column is used to exclude flagged data before any ranges are determined, but you can set useflags=False to include flagged data in the range. However, if you average in frequency, flagging will still be applied.
You can influence the memory use and the reading speed using the blocksize argument - it specifies how big a block of data to read at once (in MB). For large datasets on machines with lots of memory you may speed things up by setting this higher than the default (10 MB).
For some items, as indicated with an § in table 1.6 (in the general description of this tool), you need to call selectinit to select a portion of the data with a unique shape prior to calling this function.
Items prefixed with corrected, model, residual or obs_residual and the imaging_weight item are not available unless your measurement set has been processed either with the imager or calibrator tools.

Arguments

Inputs		
items	Item names	
	allowed:	stringArray
	Default:	
useflags	Use the data flags	
	allowed:	bool
	Default:	true
blocksize	Set the blocksize in MB	
	allowed:	int
	Default:	10

Returns
record

Example

```
      ms.open("3C273XC1.MS")
      ms.selectinit(datadescid=0)
      ms.range(["time","uvdist","amplitude","antenna1"])
#{'amplitude': array([  2.60339398e-02,   3.38518333e+01]),
# 'antenna1': array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 11, 12, 13,
#                    14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]),
# 'time': array([  4.12162940e+09,   4.12164267e+09]),
# 'uvdist': array([   46.26912101,  3727.97385983])}

    In this example the minimum and maximum observation times,
    uvdistances, data amplitudes are returned as well as a list of all the
    antennas in the antenna1 column.

    For this dataset the selectinit function did not need to be
    called as all the data is of one shape.
```

ms.lister.html

ms.lister - Function

1.3.1 List measurement set visibilities

Description

This tool lists measurement set visibility data under a number of input selection conditions. The measurement set data columns that can be listed are: the raw data, corrected data, model data, and residual (corrected - model) data.
The output table format is dynamic. Field, Spectral Window, and Channel columns are not displayed if the column contents are uniform. For example, if "spw = '1' " is specified, the spw column will not be displayed. When a column is not displayed, a message is sent to the logger and terminal indicating that the column values are uniform and listing the uniform value.
Table column descriptions:

Column Name & Description
Date/Time & Average date and time of data sample interval
Intrf & Interferometer baseline (antenna names)
UVDist & uv-distance (units of wavelength)
Fld & Field ID
SpW & Spectral Window ID
Chn & Channel number
(Correlated & Correlated polarizations (eg: RR, LL, XY)
polarization) & Sub-columns are: Amp, Phs, Wt, F
Amp & Visibility amplitude
Phs & Visibility phase
Wt & Weight of visibility measurement
F & Flag: 'F' = flagged datum; ' ' = unflagged

Arguments

Inputs		
options	Output options (not yet implemented)	
	allowed:	string
	Default:	
datacolumn	Column to list: data, model, corrected, residual	
	allowed:	string
	Default:	data
field	Fields	
	allowed:	string
	Default:	
spw	Spectral Windows	
	allowed:	string
	Default:	
antenna	Antenna/Baselines	
	allowed:	string
	Default:	
timerange	Time range	
	allowed:	string
	Default:	
correlation	Polarization correlations	
	allowed:	string
	Default:	
scan	Scan	
	allowed:	string
	Default:	
feed	Feed (not yet implemented)	
	allowed:	string
	Default:	
array	Array	
	allowed:	string
	Default:	
observation	Select by observation ID(s)	
	allowed:	string
	Default:	
uvrange	uv-distance (output units: wavelength)	
	allowed:	string
	Default:	
average	Average mode (not yet implemented)	
	allowed:	string
	Default:	
showflags	Showflags (not yet implemented)	
	allowed:	bool
	Default:	false
msselect	TaQL expression	
	allowed:	string
	Default:	
pagerows	Rows per page	
	allowed:	int
	Default:	50
listfile	Output file	
	allowed:	string
	Default:	

Returns
bool

Example

```
ms.open('AZ136.ms')
ms.lister()
```

These commands yeild the following listing:

```
Date/Time:                          RR:                      RL:                      LR:
2001/12/01/  Intrf UVDist Fld SpW      Amp   Phs       Wt F     Amp   Phs       Wt F     Amp
------------|-----|------|---|---|-------------------------|------------------------|-------
  19:30:05.0  0- 1   1400   0   0:   0.002-102.7   229035 F   0.003-178.3   239694 F   0.001
  19:30:05.0  0- 2   7203   0   0:   0.002 127.3   267464 F   0.001 165.0   305192 F   0.003
  19:30:05.0  0- 3   9621   0   0:   0.002 -55.9   179652 F   0.002 -27.1   230130 F   0.001
  19:30:05.0  0- 4   1656   0   0:   0.001 133.3   199677 F   0.002  80.6   258140 F   0.001
  19:30:05.0  0- 5   3084   0   0:   0.002 -18.4   197565 F   0.001 -83.1   228541 F   0.002
  19:30:05.0  0- 6   5020   0   0:   0.001-173.2   236475 F   0.002-104.0   257575 F   0.000
  19:30:05.0  0- 7  12266   0   0:   0.003 -34.6   264977 F   0.002   5.3   280113 F   0.001
    .
    .
    .
```

Notice that the channel column is not displayed. This measurement set contains only one channel; since the channel column values are uniform, the channel column is not displayed. Instead, message "All selected data has CHANNEL = 0" is sent to the console.

---

ms.metadata.html

ms.metadata - Function

1.3.1 Get the MS metadata associated with this MS.

Description

Get the MS metadata associated with this MS.

Arguments

Inputs		
cachesize	Maximum cache size, in megabytes, to use.	
	allowed:	float
	Default:	50

Returns

msmetadata

Example

```
# get the number of spectral windows in the specified MS
ms.open"my.ms")
metadata = ms.metadata()
ms.done()
nspw = metadata.nspw()
metadata.done()
```

---

ms.selectinit.html

ms.selectinit - Function

1.3.1 Initialize the selection of an ms

Description

A measurement set can contain data with a variety of different shapes (as described in the overall description to this tool). To allow functions to return data in fixed shape arrays you need to select, using this function, rows that contain the same data shape. You do not need to use this function if all the data in your measurement set has only one shape.
The DATA_DESC_ID column in the measurement set contains a value that maps to a particular row in the POLARIZATION and
SPECTRAL_WINDOW subtables. Hence all rows with the same value in the DATA_DESC_ID column must have the same data shape. To select all the data where the DATA_DESC_ID value is N you call this function with the datadescid argument set to N.
It is possible to have a measurement set with differing values in the DATA_DESC_ID column but where all the data is a fixed shape. For example this will occur if the reference frequency changes but the number of spectral channels is fixed. In cases like this all the data can be selected, using this function with an argument of zero. If the data shape does change and you call this function with an datadescid set to zero the return value will be False. In all other cases it will be True.
To return to the completely unselected measurement set, set the reset argument to True. This will allow you to access the full range of rows in the measurement set, rather than just the selected measurement set.
The datadescid must always be a non-negative integer.

Arguments

Inputs		
datadescid	Data description id	
	allowed:	int
	Default:	0
reset	Reset to unselected state	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
ms.open("3C273XC1.MS")
ms.selectinit(datadescid=0)
print ms.range(["uvdist"])
ms.selectinit(reset=True)
print ms.range("uvdist")
```

In this example we display the range of uv distances for the data in the specified measurement set. The first print statement will only use data where the DATA_DESC_ID column is 0. This will correspond to a specific spectral window and polarization setup. The second print statement will print the range of uv distances for all the data in the measurement set (which is the same in this case).

---

ms.msselect.html

ms.msselect - Function

1.3.1 Use the MSSelection module for data selection.

Description

A retun value of True implies that the combination of all selection expressions resulted in a non-Null combined TaQL expression. False implies that the combined TaQL could not be formed (i.e. it is Null, and the "selected MS" will be the same as the input MS).
The details of selection expressions are desribed in the MSSelection Memo.
Note that this function can be called multiple times but the result is cumulative. I.e. selection will work on the data already selected from all previous calls of this function. Use the function reset() to reset all selections to NULL (original database).

Arguments

Inputs	
items	Record with fields contain the selection experessions. Keys recognized in the record are: "spw", "time", "field", "baseline", "scan", "scanintent", "polarization", "observation", "array", "uvdist" and "taql". allowed: record Default:
onlyparse	If set to True, expressions will only be parsed but not applied to the MS for selection. When set to False, a selected MS will also be generated internally. Default is False. When only parsing is requested, the selected-MS is the same as the original MS. allowed: bool Default: false

Returns
bool

Example

```
CASA: staql={'field':'3C286', 'spw':'0~7:10~55'};
CASA: ms.open(MSNAME);
CASA: ms.msselect(staql, onlyparse=True); # For only getting the list
                                          # of indices correspoding to the selection
CASA: ndx=ms.msselectedindices();
CASA: ndx['field']
  Out[5]: array([1], dtype=int32)
   :
   :
CASA: ms.msselect(staql);  # To do the actual selection.
                           # From this point on, the ms-tool is
                           # attached to the selected MS.
```

---

ms.msselectedindices.html

ms.msselectedindices - Function

1.3.1 Return the selected indices of the MS database. The keys in the record are the same as those used in msselect function (i.e. 'spw', 'time', 'field', 'baseline', 'scan', 'scanintent', 'polarization' and 'uvdist').

Description

The return indices are the result of parsing the MSSelection expressions provided in the msselect function.

Arguments

Inputs

Returns
record

Example

---

ms.select.html

ms.select - Function

1.3.1 Select a subset of the measurement set.

Description

This function will select a subset of the current measurement set based on the range of values for each field in the input record. The range function will return a record that can be altered and used as the argument for this function. A successful selection returns True. Allowable fields are tabulated in table∼1.6 (in the general description of this tool). Unrecognized fields are ignored.
You need to call selectinit before calling this function. If you haven't then selectinit will be called for you with default arguments.
Repeated use of this function, with different arguments, will further refine the selection, resulting in a successively smaller selected measurement set. If the selected measurement set does not contain any rows then this function will return False and send a warning message in the logger. Otherwise this function will return True. To undo all the selections you need to use the selectinit function (with reset=True).

Arguments

Inputs		
items	record with fields contain ranges and enumerations	
	allowed:	record
	Default:	

Returns

bool

Example

```
ms.open("3C273XC1.MS")
ms.selectinit(datadescid=0)
ms.select({'antenna1':[1,3,5],'uvdist':[1200.,1900.]})
```

```
ms.select({'time':[4121629420.,4121638290.]})
start = qa.getvalue(qa.convert(qa.quantity('1989/06/27/01:03:40'),'s'))
stop = qa.getvalue(qa.convert(qa.quantity('1989/06/27/03:31:30'),'s'))
rec = {}
rec['time'] = [start, stop]
ms.select(items=rec)
```

This example selects all the data from the measurement set where the value in the DATA_DESC_ID column is zero. This corresponds to a particular spectral window and polarization setup. It then selects all the data where the first antenna in the interferometer is number one, three or five and where the uv distance is between 1200 and 1900 meters. Finally it selects all the data which was observed between 4121629420 seconds and 4121638290 seconds (since zero hours on the day where the modified Julian day is zero). Since this time in seconds is quite obscure I have also illustrated how to use the quanta tool to convert a date/time string into seconds which can then be used to perform the same time selection.

The selections are cumulative so that at the end of this example only data in the specified time range, with the specified, interferometers, uv distances, spectral window and polarization setup are selected.

---

ms.selecttaql.html

ms.selecttaql - Function

1.3.1 Select a subset of the measurement set.

Description

This function will select a subset of the current measurement set based on the standard TaQL selection string given.
Repeated use of this function, with different arguments, will further refine the selection, resulting in a successively smaller selected measurement set. If the selected measurement set does not contain any rows then this function will return False and send a warning message in the logger. Otherwise this function will return True. To undo all the selections you need to use the selectinit function (with reset=True). Note that index values used in the TaQL string are zero-based as are all tool indices.

Arguments

Inputs		
msselect	TaQL selection string	
	allowed:	string
	Default:	

Returns
bool

Example

```
ms.open("3C273XC1.MS")
ms.selectinit(datadescid=0)
ms.select({'antenna1':[0,2,4],'uvdist':[1200.,1900.]})
ms.selecttaql('ANTENNA1==2')
ms.range(["ANTENNA1","ANTENNA2"])
# {'antenna1': array([2]),
```

```
  #  'antenna2': array([ 6,  9, 11, 18, 20, 21, 24])}
```

This example selects all the data from the measurement set where the value in the DATA_DESC_ID column is zero. This corresponds to a particular spectral window and polarization setup. It then selects all the data where the first antenna in the interferometer is number zero, two or four and where the uv distance is between 1200 and 1900 meters. Finally it uses a query to select all the data for which the ANTENNA1 column is 2 (this selects the middle antenna of the previous, zero-based, selection). The selections are cumulative so that at the end of this example only data in the specified time range, with the specified, interferometers, uv distances, spectral window and polarization setup are selected.

---

ms.selectchannel.html

ms.selectchannel - Function

1.3.1 Select and average frequency channels

Description

This function allows you to select a subset of the frequency channels in the current measurement set. This function can also average, over frequency channels, prior to providing the values to the user.
Selection on channels is not allowed using either the select or command functions as they can only select entire rows in a measurement set. Channel selection involves accessing only some of the values in a row. Like all the selection functions this function does not change the current measurement but updates the measurement set selection parameters so that functions like getdata will return the desired subset of the data. Repeated use of this function will overwrite any previous channel selection.
There are four parameters, the number of output channels, the first input channel to use, the number of input channels to average into one output channel, and the increment in the input spectrum for the next output channel. All four parameters need to be specified.
This function return True if the selection was successful, and False if not. In the latter case an error message will also be sent to the logger.
You need to call selectinit before calling this function. If you haven't then selectinit will be called for you with default arguments.

Arguments

Inputs		
nchan	Number of output channels, positive integer	
	allowed:	int
	Default:	1
start	First input channel to use, positive integer	
	allowed:	int
	Default:	0
width	Number of input channels to average together, positive integer	
	allowed:	int
	Default:	1
inc	Increment to next (group of) input channel(s), positive integer	
	allowed:	int
	Default:	1

Returns
bool

Example

```
ms.fromfits("NGC5921.MS",
            "/usr/lib/casapy/data/demo/NGC5921.fits")
ms.selectinit(datadescid=0)
ms.selectchannel(3,2,5,3)
```

This example selects all the data from the measurement set where the value in the DATA_DESC_ID column is zero. This corresponds to a particular spectral window and polarization setup. It then selects on frequency channels to produce 3 output channels, the first output channel is the average of channels 2,3,4,5,6 in the input, the second output channel is the average of channel 5,6,7,8,9 and the third is the average of channels 8,9,10,11,12.

ms.selectpolarization.html

ms.selectpolarization - Function

1.3.1 Selection and convertion of polarizations

Description

This function allows you to select a subset of the polarizations in the current measurement set. This function can also setup conversion to different polarization representations.
You specify the polarizations using a string vector. Allowable strings are include I, Q, U, V, RR, RL, LR, LL, XX, YY, XY, YX. These string must be specified in upper case. If the polarizations match those present in the measurement set they will be selected directly, otherwise all polarizations are read and then a conversion step is done. If the conversion cannot be done then an error will be produced when you try to access the data.
This function return True if the selection was successful, and False if not.
You need to call selectinit before calling this function. If you haven't then selectinit will be called for you with default arguments.

Arguments

Inputs		
wantedpol	The polarizations wanted	
	allowed:	stringArray
	Default:	

Returns
bool

Example

```
ms.open("3C273XC1.MS")
ms.selectinit(datadescid=0)
ms.selectpolarization(["I","V"])
```

```
  rec = ms.getdata("data")
```

This example selects all the data from the measurement set where the value in the DATA_DESC_ID column is zero. This corresponds to a particular spectral window and polarization setup. It then selects the I and V polarizations and when the getdata function is called the conversion from RR, LL, LR, RL polarizations to I and V occurs.

---

ms.regridspw.html

ms.regridspw - Function

1.3.1 transform spectral data to different reference frame and/or regrid the frequency channels

Description

This function permits you to transform the spectral data of your measurement set to a given reference frame. The present reference frame information in the MS is examined and the transformation performed accordingly. Since all such transformations are linear in frequency, a pure change of reference frame only affects the channel boundary definitions.
In addition, the function permits you to permanently regrid the data, i.e. reduce the channel number and/or move the boundaries using several interpolation methods (selected using parameter "interpolation"). The new channels are equidistant in frequency (if parameter "mode" is chosen to be vrad or freq, or equidistant in wavelength if parameter "mode" is chosen to be vopt or wave). If "mode" is chosen to be "chan", the regridding is performed by combining the existing channels, i.e. not moving but just eliminating channel boundaries where necessary.
The regridding is applied to the channel definition and all data of the MS, i.e. all columns which contain arrays whose dimensions depend on the number of channels. The input parameters are verified before any modification is made to the MS.
The target reference frame can be set by providing the name of a standard reference frame (LSRK, LSRD, BARY, GALACTO, LGROUP, CMB, TOPO, GEO, or SOURCE, default = no change of frame) in parameter "outframe". For each field in the MS, the channel frequencies are transformed from their present reference frame to the one given by parameter "outframe".
If the regridding parameters are set, they are interpreted in the "outframe" reference frame. The regridding is applied to the data after the reference frame transformation.

Arguments

Inputs	
outframe	Name of the reference frame to transform to (LSRK, LSRD, BARY, GALACTO, LGROUP, CMB, GEO, TOPO, or SOURCE). SOURCE is meant for solar system work and corresponds to GEO + a radial velocity correction (only available for ephemeris objects). If no reference frame is given, the present reference frame given by the data is used, i.e. the reference frame is not changed. The observatory position is taken as the average of all antenna positions. allowed: string Default: LSRK
mode	The quantity (radio velocity (m/s), optical velocity (m/s), frequency (Hz), wavelength (m), or original channels) in which the user would like to give the regridding parameters below ("center", "chanwidth", "bandwidth"): vrad, vopt, freq, wave, or chan. allowed: string Default: chan
restfreq	Required in case the value of mode is "vrad" or "vopt": Rest frequency (Hz) for the conversion of the regrid parameters "center", "chanwidth", and "bandwidth" to frequencies. allowed: double Default: -3E30
interpolation	Name of the interpolation method (NEAREST, LINEAR, SPLINE, CUBIC, FFTSHIFT) used in the regridding. Flagging information is combined using "inclusive or". allowed: string Default: LINEAR
start	Desired lower edge of the spectral window after regridding in the units given by "mode" and in the reference frame given by "outframe". If no value is given, it is determined from "center" and "bandwidth". allowed: double Default: -3E30
center	(Alternative to setting the parameter "start".) Desired center of the spectral window after regridding in the units given by "mode" and in the reference frame given by "outframe". If no value is given, the center is determined from "start" and "bandwidth" or, if "start" is not given either, it is kept as it is. allowed: double Default: -3E30
bandwidth	Desired width of the entire spectral window after regridding in the units given by "mode" and in the reference frame given by "outframe". If no value is given or the given width is larger than the bandwidth of the data, the width will be truncated to the maximum width possible symmetrically around the value given by "center". allowed: double Default: -1.
chanwidth	Desired width of the channels in the units given by "mode" and in the reference frame given by "outframe". This implies that channels will be equidistant in the unit

Returns
bool

Example

```
  ms.fromfits("NGC5921.MS", "/usr/lib/casapy/data/demo/NGC5921.fits")
  ms.regridspw(outframe="LSRK")

This example reads a measurement set and transforms its spectral axis to
the LSRK reference frame.

  ms.regridspw(outframe="BARY", mode="vrad",
  center=73961800., chanwidth=50., bandwidth=1000.,
  restfreq=1420405750e6)

In this example, all spectral windows in the MS will be transformed
to the BARY reference frame and then be regridded such that the center of the
new spectral window is at radio velocity = 73961800. m/s (BARY). If the bandwidth
of the observation is large enough the total width of the spectral window will
be 1000 m/s, i.e. 20 channels of width 50 m/s, 10 on each side of the given
center.

  ms.regridspw(mode="vopt", restfreq=1420405750e6)

In this example the channels are regridded such that they are
equidistant in optical velocity. The reference frame and number of channels
is kept as is.

  ms.regridspw(mode="chan", center=64, chanwidth=2,
  bandwidth=102)

In this example, the channels are regridded such that the new bandwidth is
102 of the original channels centered on the original channel 64, and the new
channels are twice as wide as the original channels.
```

---

ms.cvel.html

ms.cvel - Function

1.3.1 transform spectral data to different reference frame and/or regrid the frequency channels

Description

This function permits you to transform the spectral data of your measurement set to a given reference frame and/or regrid it. It will combine all spectral windows of the MS into one.

Arguments

Inputs	
mode	"channel", "velocity", "frequency", or "channel_b", default = "channel" allowed: string Default: channel
nchan	number of channels, default = -1 = all allowed: int Default: -1
start	start channel, default = 0 allowed: any Default: variant 0
width	new channel width, default = 1 allowed: any Default: variant 1
interp	interpolation method "nearest", "linear", "spline", "cubic", "fftshift", default = linear allowed: string Default: linear
phasec	phase center, default = first field allowed: any Default: variant
restfreq	rest frequency, default = 1.4GHz allowed: any Default: variant 1.4GHz
outframe	LSRK, LSRD, BARY, GALACTO, LGROUP, CMB, GEO, TOPO, or SOURCE default = "" = keep reference frame. allowed: string Default:
veltype	radio or optical, default = radio allowed: string Default: radio
hanning	If true, perform hanning smoothing before regridding. allowed: bool Default: true

Returns
bool

Example

ms.cvelfreqs.html

ms.cvelfreqs - Function

1.3.1 calculate the transformed grid of the SPW obtained by combining a given set of SPWs (MS is not modified)

Description

Take the spectral grid of a given spectral window, tranform and regrid it as prescribed by the given grid parameters (same as in cvel and clean) and return the transformed values as a list. The MS is not modified. Useful for tests of gridding parameters before using them in cvel or clean.

Arguments

Inputs	
spwids	The list of ids of the spectral windows from which the input grid is to be taken. allowed: intArray Default: 0
fieldids	The list of ids of the fields which are selected (for observation time determination), default: all allowed: intArray Default: 0
obstime	the observation time to assume, default: time of the first row of the MS = allowed: string Default:
mode	"channel", "velocity", "frequency", or "channel_b", default = allowed: string Default: channel
nchan	number of channels, default = all = allowed: int Default: -1
start	start channel, default = allowed: any Default: variant 0
width	new channel width, default = allowed: any Default: variant 1
phasec	phase center, default = first field in selection = allowed: any Default: variant
restfreq	rest frequency, default = allowed: any Default: variant 1.4GHz
outframe	LSRK, LSRD, BARY, GALACTO, LGROUP, CMB, GEO, TOPO, or SOURCE default = keep reference frame = allowed: string Default:
veltype	radio or optical, default = allowed: string Default: radio
verbose	If true, create log output allowed: bool Default: true

Returns
doubleArray

Example

```
ms.open('my.ms')
ms.cvelfreqs(spwids=[1], mode='channel', nchan=20, start=2, width=3, outframe='LSRK')

will take the grid of SPW 1 (i.e. the second in the SPW table), regrid it as in cvel
with the given grid parameters and return the resulting channel centers as an array.
The MS is not modified. See help cvel for more details on the grid parameters.
```

---

ms.getdata.html

ms.getdata - Function

1.3.1 Read values from the measurement set.

Description

This function will read the specified items from the currently selected measurement set and returns them in fields of a record. The main difference between this and direct access of the table, using the table tool, is that this function reads data from the *selected* measurement set, it provides access to derived quantities like amplitude & flag_sum and it can reorder the data.
The items to read are specified, as with the range function, using a vector of strings. Table∼1.6 shows the allowable items. Unrecognized items will result in a warning being sent to the logger. Duplicate items are silently ignored.
The record that is returned contains fields that correspond to each of the specified items. Most fields will contain an array. The array may be one, two or three dimensional depending on whether the corresponding row in the measurement set is a scalar, one or two dimensional. Unless the ifraxis argument is set to T the length of the last axis on these arrays will correspond to the number of rows in the selected measurement set.
If the ifraxis argument is set to True, the row axis is split into an interferometer axis and a time axis. For example a measurement set with 90 rows, in an array with 6 telescopes (so that there are 15 interferometers), may have a data array of shape `[4,32,90]` if ifraxis is False or `[4,32,15,6]`, if ifraxis is True (assuming there are 4 correlations and 32 channels). If there are missing rows as will happen if not all interferometers where used for all time-slots then a default value will be inserted.
This splitting of the row axis may not happen for items where there is only a single value per row. For some items the returned vector will contain only as many values as there are interferometers and it is implicit that the same value should be used for all time slots. The antenna1, antenna2, feed1, feed2 & ifr_number items fall in this category. For other items the returned vector will have as many values as there are time slots and it is implicit that the same value should be used for all interefometers. The field_id, scan_number, data_desc_id & time items fall into this category.
The axis_info item provides data labelling information. It returns a record with the following fields: corr_axis, freq_axis, ifr_axis & time_axis. The latter two fields are not present if ifr_axis is set to False. The corr_axis field contains a string vector with elements like 'RR' or 'XY' that indicates which polarizations where correlated together to produce the data. The length of this vector will always be the same as the length of the first axis of the data

array. The freq_axis field contains a record with two fields, chan_freq & resolution. Each of these fields contains vectors which indicate the centre frequency and spectral resolution (FWHM) of each channel. The length of these vectors will be the same as the length of the second axis in the data. The ifr_axis field contains fields: ifr_number, ifr_name, ifr_shortname & baseline. The ifr_number is the same as returned by the ifr_item, the ifr_name & ifr_shortname are string vecors containing descriptions of the interferometer and the baseline is the Euclidian distance, in meters between the two antennas. All of these vectors have a length equal to the number of interferometers in the selected measurement set ie., to the length of the third axis in the data when ifraxis is True. The time_axis field contains the MJD seconds field and optionally the HA, UT & LAST fields. To include the optional fields you need to add the ha, last or ut strings to the list of requested items. All the fields in the time_axis record contain vectors that indicate the time at the midpoint of the observation and are in seconds. The MJD seconds field is since 0 hours on the day having a modified julian day number of zero and the rest are since midnight prior to the start of the observation.
An optional gap size can be specified to visually separate groups of interferometers with the same antenna1 index (handy for identifying antennas in an interferometer vs time display). The default is no gap.
An optional increment can be specified to return data from every row matching the increment only.
When the average flag is set, the data will be averaged over the time axis if the ifraxis is True or the row axis i.e., different interferometers and times may be averaged together. In the latter case, some of the coordinate information, like antenna_id, will no longer make sense.
You need to call selectinit before calling this function. If you haven't then selectinit will be called for you with default arguments.
Items prefixed with either; corrected, model, residual or obs_residual and the imaging_weight item are not available unless your measurement set has been processed either with the imager or calibrator tools.

Arguments

Inputs		
items	Item names	
	allowed:	stringArray
	Default:	
ifraxis	Create interferometer axis if True	
	allowed:	bool
	Default:	false
ifraxisgap	Gap size on ifr axis when antenna1 changes	
	allowed:	int
	Default:	0
increment	Row increment for data access	
	allowed:	int
	Default:	1
average	Average the data in time or over rows	
	allowed:	bool
	Default:	false

Returns
record

Example

```
ms.open("3C273XC1.MS")
ms.selectinit(datadescid=0)
# The following line causes an EXCEPTION
# d = ms.getdata(["amplitude","axis_info","ha"],ifraxis=True)
# so we settle for MJDseconds rather than HA in seconds
d = ms.getdata(["amplitude","axis_info"],ifraxis=True)
tstart = min(d["axis_info"]["time_axis"]["MJDseconds"])
tstop  = max(d["axis_info"]["time_axis"]["MJDseconds"])
maxamp = max(max(d["amplitude"][:,0,0,0]),max(d["amplitude"][0,:,0,0]),
    max(d["amplitude"][0,0,:,0]),max(d["amplitude"][0,0,0,:]))
print "MJD start time (seconds) =", tstart
# MJD start time (seconds) = 4121629400.0
print "MJD stop time (seconds) =", tstop
# MJD stop time (seconds) = 4121642670.0
# MJDseconds Correlation amplitude
print "Maximum correlation amplitude =", maxamp
# Maximum correlation amplitude = 33.5794372559
chan = 0
```

```
corr = 0
freqGHz = d["axis_info"]["freq_axis"]["chan_freq"][chan]/1.0E9
baselineStr = d["axis_info"]["ifr_axis"]["ifr_name"][corr]
corrStr = d["axis_info"]["corr_axis"][corr]
tcoord = d["axis_info"]["time_axis"]["MJDseconds"]
acoord = d["amplitude"][0,0,0,:]
print "Frequency", freqGHz, "GHz", "Baseline", baselineStr, "(", corrStr, ")"
print "MJDseconds", "Correlation amplitude"
for i in range(len(tcoord)):
    print tcoord[i], acoord[i]
#
# Frequency [ 8.085] GHz Baseline 1-2 ( RR )
# MJDseconds Correlation amplitude
# 4121629400.0 29.2170944214
# 4121629410.0 29.1688995361
# 4121629420.0 29.2497825623
# 4121629430.0 29.2029647827
# 4121629440.0 29.166015625
# 4121629450.0 29.2417526245
# 4121629460.0 29.2867794037
# 4121638270.0 0.0
# 4121638280.0 29.4539775848
# 4121638290.0 29.472661972
# 4121638300.0 29.4424362183
# 4121638310.0 29.4234466553
# 4121638320.0 29.4018745422
# 4121638330.0 29.3326053619
# 4121638340.0 29.3575496674
# 4121642600.0 31.1411132812
# 4121642610.0 31.0726108551
# 4121642620.0 31.1242599487
# 4121642630.0 31.0505466461
# 4121642640.0 31.0448284149
# 4121642650.0 30.9974422455
# 4121642660.0 31.0648326874
# 4121642670.0 31.0638961792
```

This example selects all the data from the measurement set where the value in the DATA_DESC_ID column is zero. This corresponds to a particular spectral window and polarization setup. It then gets the correlated amplitude, and the axis information from this selected measurement set. This is returned in the casapy variable d. The remainder of the example prints a table of 'hour angle' and corresponding 'correlated amplitude' for the first channel, correlation and baseline.

ms.putdata.html

ms.putdata - Function

1.3.1 Write new values into the measurement set

Description

This function allows you to write values from casapy variables back into the measurement set table. The main difference between this and directly accessing the table using the table tool is that this function writes data to the *selected* measurement set.
Unlike the getdata function you can only put items that correspond to actual table columns. You cannot change the data shape either so that the number of correlations, channels and rows (or intereferometers/time slots) must match the values in the selected measurement set. If the values were obtained using the getdata function with ifraxis argument set to True, then any default values added to fill in missing intereferometer/timeslots pairs will be ignored when writing the modified values back using this function.
The measurement set has to be opened for read/write access to be able to use this function.
You need to call selectinit before calling this function. If you haven't then selectinit will be called for you with default arguments.
Items prefixed with either; corrected, model, residual or obs_residual and the imaging_weight item are not available unless your measurement set has been processed either with the imager or calibrator tools.

Arguments

Inputs		
items	Record with items and their new values	
	allowed:	record
	Default:	

Returns
bool

Example

```
ms.open("3C273XC1.MS", nomodify=False)
ms.selectinit(datadescid=0)
rec = ms.getdata(["weight","data"])
rec['weight'][:,:] = 1
import Numeric
meanrec = Numeric.average(rec['data'],axis=None)
print "Mean data value = ", meanrec
rec['data'][:,:,:] -= meanrec
ms.putdata(rec)
```

This example selects all the data from the measurement set where the value in the DATA_DESC_ID column is zero. This corresponds to a particular spectral window and polarization setup. Note that the measurement set was opened for writing as well as reading. The third line reads all the weights and the data into the casapy variable rec. The weights are set to one. The more obscure syntax is used as typing rec['weight'] = 1 will not preserve the shape of the weight array. The data then has its mean subtracted from it. The average function is defined in Numeric module. Finally the data is written back into the measurement set table. (NOTE: normally one should not modify the raw data column. Such adjustments are more appropriate for the corrected_data column, if it exists.)

---

ms.concatenate.html

ms.concatenate - Function

1.3.1 Concatenate two measurement sets

Description

This function concatenates two measurement sets together.
The data is copied from the measurement set specified in the msfile arguement to the end of the measurement set attached to the ms tool. If a lot of data needs to be copied this function may take some time. You need to open the measurement set for writing in order to use this function.

Arguments

Inputs	
msfile	The name of the measurement set to append allowed: string Default:
freqtol	Frequency difference within which 2 spectral windows are considered similar; e.g '10Hz' allowed: any Default: variant 1Hz
dirtol	Direction difference within which 2 fields are considered the same; e.g '1mas' allowed: any Default: variant 1mas
weightscale	Scale the weights of the MS to be appended by this factor allowed: float Default: 1.
handling	Switch for the handling of the Main and Pointing tables: 0=standard, 1=no Main, 2=no Pointing, 3=no Main and Pointing, 4=virtual allowed: int Default: 0
destmsfile	Optional support for virtual concat: empty table (no subtables) where to store the appended MS copy allowed: string Default:
respectname	If true, fields with a different name are not merged even if their direction agrees allowed: bool Default: false

Returns
bool

Example

```
ms.open("3C273XC1.MS", nomodify=False)
ms.concatenate("BLLAC.ms", '1GHz', '1arcsec')
ms.done()
```

This example appends the data from the BLLAC measurement set to

```
the end of the 3C273 measurement set. Its going to assume a
frequency tolerance of 1GHz and position tolerance of 1 arcsec in
deciding if the spw and field in the measurementsets are
similar or not.
```

---

ms.testconcatenate.html

ms.testconcatenate - Function

1.3.1 Concatenate only the subtables of two measurement sets excluding the POINTING table (resulting MAIN and POINTING table not useful)

Description

This function acts like ms.concatenate() with handling==3 (do not concatenate the MAIN and POINTING tables). This is useful for generating, e.g., SPECTRAL_WINDOW and FIELD tables which contain all used SPW and FIELD ids for a set of MSs without having to actually carry out a time-consuming concatenation on disk. The MAIN table in the resulting output MS is that of the original MS, i.e. it is not touched.

Arguments

Inputs	
msfile	The name of the measurement set from which the subtables should be appended allowed: string Default:
freqtol	Frequency difference within which 2 spectral windows are considered similar; e.g '10Hz' allowed: any Default: variant 1Hz
dirtol	Direction difference within which 2 fields are considered the same; e.g '1mas' allowed: any Default: variant 1mas
respectname	If true, fields with a different name are not merged even if their direction agrees allowed: bool Default: false

Returns
bool

Example

```
tb.open("3C273XC1.MS")
tb.copy("TEMP.MS", norows=True)
tb.close()
ms.open("TEMP.MS", nomodify=False)
ms.testconcatenate("3C273XC1.ms", '1GHz', '1arcsec')
ms.testconcatenate("BLLAC.ms", '1GHz', '1arcsec')
ms.done()
```

This example makes a copy of the structure of an MS and then
appends the subtables data from two measurement sets to the empty structure.
Its going to assume a frequency tolerance of 1GHz and position tolerance
of 1 arcsec in deciding if the spw and field in the measurementsets are
similar or not.

---

ms.virtconcatenate.html

ms.virtconcatenate - Function

1.3.1 Concatenate two measurement sets virtually

Description

This function virtually concatenates two measurement sets together such that they can later be turned into a multi-MS with createmultims().
You need to open the measurement set for writing in order to use this function.

Arguments

Inputs	
msfile	The name of the measurement set to append allowed: string Default:
auxfilename	The name of a auxiliary file which is needed when more than two MSs are to be concatenated. allowed: string Default:
freqtol	Frequency difference within which 2 spectral windows are considered similar; e.g '10Hz' allowed: any Default: variant 1Hz
dirtol	Direction difference within which 2 fields are considered the same; e.g '1mas' allowed: any Default: variant 1mas
weightscale	Scale the weights of the MS to be appended by this factor allowed: float Default: 1.
respectname	If true, fields with a different name are not merged even if their direction agrees allowed: bool Default: true

Returns

bool

Example

```
ms.open("3C273XC1.ms", nomodify=False)
ms.virtconcatenate("3C273XC1-2.ms", '3Caux.dat', '1GHz', '1arcsec')
ms.virtconcatenate("3C273XC1-3.ms", '3Caux.dat', '1GHz', '1arcsec')
ms.close()
os.remove('3Caux.dat')
m.createmultitms(concatvis,
                ["3C273XC1.ms","3C273XC1-2.ms","3C273XC1-3.ms"],
[],
True, # nomodify
False,# lock
True) # copysubtables from first to all other members
ms.close()
```

This example virtually appends the data from the 3C273XC1-2 and 3C273XC1-3 to the end of the 3C273XC1 measurement set. Its going to assume a frequency tolerance of 1GHz and position tolerance of 1 arcsec in deciding if the spw and field in the measurementsets are similar or not. The file 3Caux.dat which is created in the process is no longer needed after the last call to virtconcatenate() and can be deleted.

---

ms.timesort.html

ms.timesort - Function

1.3.1 Sort the main table of an MS by time

Description

This function sorts the main table of the measurement set by the contents of the column TIME in ascending order and writes a copy of the MS with the sorted main table into newmsfile.
If no newmsname is given, a sorted copy of the MS is written into a new MS under the name x.sorted (where x is the name of the original MS). The original MS is then closed and deleted. The new MS is renamed to the name of the original MS and then reopened.

Arguments

Inputs	
newmsname	Name of the output measurement set (default: overwrite original) allowed: string Default:

Returns
bool

Example

```
ms.open("3C273XC1.MS", nomodify=False)
ms.timesort()
ms.done()

This example sorts the main table of 3C273XC1.MS by time.
The original MS is overwritten by the sorted one.
```

ms.sort.html

ms.sort - Function

1.3.1 Sort the main table of an MS using a custom set of columns

Description

This function sorts the main table of the measurement set by the contents of the input set of columns in ascending order and writes a copy of the MS with the sorted main table into newmsfile.
If no newmsname is given, a sorted copy of the MS is written into a new MS under the name x.sorted (where x is the name of the original MS). The original MS is then closed and deleted. The new MS is renamed to the name of the original MS and then reopened.

Arguments

Inputs		
newmsname	Name of the output measurement set (default: overwrite original)	
	allowed:	string
	Default:	
columns	Vector of column names (case sensitive).	
	allowed:	stringArray
	Default:	

Returns
bool

Example

```
ms.open("3C273XC1.MS", nomodify=False)
ms.sort(['ANTENNA1','ANTENNA2'])
ms.done()
```

This example sorts the main table of 3C273XC1.MS by ANTENNA1 and then ANTENNA2. The original MS is overwritten by the sorted one.

---

ms.contsub.html

ms.contsub - Function

1.3.1 Subtract the continuum from the visibilities

Description

NOT FULLY IMPLEMENTED YET. uvcontsub uses the cb tool for now. (The only reason to implement it in ms is to save time and disk space.) This function estimates the continuum emission of the MS and writes a MS with that estimate subtracted, using the ms tool. The estimate is made, separately for the real and imaginary parts of each baseline, by fitting a low order polynomial to the unflagged visibilities selected by fitspw (depending on combine).

Arguments

Inputs	
outputms	The name of the resulting measurement set allowed: string Default:
fitspw	Line-free spectral windows (and :channels) to fit to allowed: variant Default: *
fitorder	The order of the polynomial to use when fitting. allowed: int Default: 1
combine	Ignore changes in these columns (spw, scan, and/or state) when fitting. allowed: string Default:
spw	Spectral windows (and :channels) to select allowed: variant Default: *
unionspw	The union of fitspw and spw, i.e. how much needs to be read. '*' always works, but may be more than you need. allowed: variant Default: *
field	Fields to include, by names or 0-based ids. (" => all) allowed: variant Default:
scan	Only use the scan numbers requested using the msselection syntax. allowed: variant Default:
intent	Only use the requested scan intents. allowed: string Default:
correlation	Limit data to specific correlations (LL, XX, LR, XY, etc.). allowed: string Default:
obs	Only use the requested observation IDs. allowed: string Default:
whichcol	'DATA', 'MODEL_DATA', 'CORRECTED_DATA', and/or 'FLOAT_DATA' allowed: string Default: CORRECTED_DATA

Returns
bool

Example

```
ms.open("multiwin.ms")
ms.contsub('contsub.ms', fitspw='0:0~123;145~211,2:124~255', fitorder=0,
           field=[0], spw='0,2')
```

In this example the continuum estimates are made by seperately averaging channels 0:0~123;145~211 and 2:124~255, and the separate estimates are subtracted from spws 0 and 2. The output only includes field 0 and spws 0 and 2 (now called 1).

```
ms.contsub('contsub.ms', fitspw='0:0~123;145~211,2:124~255', fitorder=0,
           field=[0], combine='spw')
ms.close()
```

This time the estimate was made by simultaneously averaging channels 0:0~123;145~211 and 2:124~255, and the continuum is subtracted from all the spws, including 1 (treated as a completely line-filled spw). The output only includes field 0.

---

ms.statwt.html

ms.statwt - Function

1.3.1 Set WEIGHT and SIGMA from the scatter of the visibilities

Description

NOT IMPLEMENTED YET.
This function estimates the noise from the scatter of the visibilities, sets SIGMA to it, and WEIGHT to SIGMA**-2.
Ideally the visibilities used to estimate the scatter, as selected by fitspw and fitcorr, should be pure noise. If you know for certain that they are, then setting dorms to True will give the best result. Otherwise, use False (standard sample standard deviation). More robust scatter estimates like the interquartile range or median absolute deviation from the median are not offered because they require sorting by value, which is not possible for complex numbers.
To beat down the noise of the noise estimate, the sample size per estimate can be made larger than a single spw and baseline. (Using combine='spw' is to interpolate between spws with line-free channels is recommended when an spw has no line-free channels.) timebin smooths the noise estimate over time. windowtype sets the type of time smoothing.
WEIGHT and SIGMA will not be changed for samples that have fewer than minsamp visibilities. Selected visibilities for which no noise estimate is made will be flagged. Note that minsamp is effectively at least 2 if dorms is False, and 1 if it is True.

Arguments

Inputs	
dorms	How the scatter should be estimated (True -> rms, False -> stddev) allowed: bool Default: false
byantenna	How the scatters are solved for (by antenna or by baseline) allowed: bool Default: true
sepacs	If solving by antenna, treat autocorrs separately allowed: bool Default: true
fitspw	Line-free spectral windows (and :channels) to get the scatter from. (" => all) allowed: variant Default: *
fitcorr	Correlations (V, LL, XX, LR, XY, etc.) to get the scatter from. (" => all) allowed: variant Default:
combine	Ignore changes in these columns (spw, scan, and/or state) when getting the scatter. allowed: string Default:
timebin	Duration of the moving window over which to estimate the scatter. Defaults to 0s, with an effective minimum of 1 integration. allowed: variant Default: 0s
minsamp	The minimum number of visibilities for a scatter estimate allowed: int Default: 3
field	Fields to reweight, by names or 0-based ids. (" => all) allowed: variant Default:
spw	Spectral windows to reweight. (" => all) allowed: variant Default: *
antenna	Select data based on antenna/baseline allowed: any Default: variant
timerange	Select data by time range allowed: string Default:
scan	Scan numbers to reweight. (" => all) allowed: variant Default:
intent	Scan intents to reweight. (" => all) allowed: string Default:
array	Select (sub)array(s) by array ID number allowed: variant Default:
correlation	Correlations (LL, XX, LR, XY, etc.) to reweight. ("

Returns
bool

Example

```
ms.open("multiwin.ms", nomodify=False)
ms.statwt(fitspw='0:0~123;145~211,2:124~255', field=[0], spw='0,2')

In this example the noise estimates are seperately made from and applied
to spws 0 and 2.

ms.statwt(fitspw='0:0~123;145~211,2:124~255', fitorder=0, field=[0],
          combine='spw')
ms.close()

This time the estimate for each baseline is made from the line-free channels
of spws 0 and 2, and applied to all the spws, including 1 (which could be a
completely line-filled spw).
```

---

ms.split.html

ms.split - Function

1.3.1 make a new ms from a subset of an existing ms, adjusting subtables and indices

Description

This function splits out part of the MS into a new MS. Time and channel averaging can be performed in the process (but not in the same call). When splitting multiple spectral windows, the parameters `nchan, start, step` can be vectors, so that each spectral window has its own selection on averaging and number of output channels. But the option of using only one value for each of these parameters means that it will be replicated for all the spectral windows selected.

Arguments

Inputs	
outputms	The name of the resulting measurement set allowed: string Default:
field	Fields to include, by names or 0-based ids. (" => all) allowed: variant Default:
spw	Spectral windows (and :channels) to select allowed: variant Default: *
step	number of input per output channels - Int vector of length 1 or same as spw allowed: intArray Default: 1
baseline	Antenna names or indices to select (" => all) allowed: variant Default:
timebin	Duration for averaging. Defaults to no averaging. allowed: variant Default: -1s
time	Only use data in the given time range, using the msselection syntax. allowed: string Default:
scan	Only use the scan numbers requested using the msselection syntax. allowed: variant Default:
uvrange	Limit data by uv distance using the msselection syntax. allowed: variant Default:
taql	For the TAQL experts, flexible data selection using the TAQL syntax allowed: string Default:
whichcol	'DATA', 'MODEL_DATA', 'CORRECTED_DATA', 'FLOAT_DATA', 'LAG_DATA', and/or 'all' allowed: string Default: DATA
tileshape	Tile shape of the disk data columns, most users should not need to touch this parameter [0] => normal tiling, [1] => fast mosaic style tile [4,15,351] => a tile shape of 4 pol 15 chan and 351 rows allowed: variant Default:
subarray	Limit data to specific (sub)array numbers. allowed: variant Default:
combine	Ignore changes in these columns (scan, and/or state) when time averaging. allowed: string Default:
correlation	Limit data to specific correlations (LL, XX, LR, XY, etc.). allowed: string Default:

Returns
bool

Example

```
ms.open("multiwin.ms")
ms.split('subms.ms', field=[0], spw=[0], nchan=[10],
         start=[0], step=[5], whichcol='CORRECTED_DATA')

In this example we split out data from the 1st field and 1st
spectral window. The output data will have 10 channels which is
taken from 50 channels from the input data starting at channel 0 and
averaging every 5.


ms.open("multiwin.ms")
ms.split('subms.ms', field=[0], spw=[0,1,2,3], nchan=[10],
         start=[0], step=[5], whichcol='CORRECTED_DATA')

In this example we split out data from the 1st field and four spectral
windows. The output data will have 4 spectral windows each of 10
channels which is taken from 50 channels from the input data starting
at channel 0 and averaging every 5.


ms.open("multiwin.ms")
ms.split('subms.ms', field=[0], spw=[0,1,2,3], nchan=[10,10,30,40],
         start=[0,4,9,9], step=[1,10,5,2], whichcol='CORRECTED_DATA')

In this example we split out data from the 1st field and four spectral
windows. There will be four spectral windows in the output data,
with 10, 10, 30 and 40 channels respectively. These are averages
of the input spectral windows. The first output spectral window
will be formed by picking 10 channels, starting at 0 with no
averaging, of the input spwid 0. The second output spectral window
will consists of 10 channels and is formed by picking 100 channels
from spwid 1 of the input data, starting at channel 4, and every
10 channels to make one output channel.


ms.open("WSRT.ms")
```

```
ms.split('subms.ms', timebin='20s', whichcol='all', combine='scan')
ms.close()
```

This example averages a WSRT MS into 20s bins, selecting whichever of DATA, MODEL_DATA, CORRECTED_DATA, or FLOAT_DATA, or LAG_DATA is present. Normally the bins would not cross scans, but in this MS the scan number goes up with each integration, making it redundant enough with time that it would defeat any time averaging. Therefore the combine parameter forces the SCAN column to be ignored for setting the bins.

---

ms.partition.html

ms.partition - Function

1.3.1 make a new ms from a subset of an existing ms, without changing any subtables

Description

This function splits out part of the MS into a new MS. Time averaging can be performed in the process. Unlike split, the subtables and IDs (ANTENNA1, DATA_DESCRIPTION_ID, etc.) are never changed to account for the selection.
As a side effect of that property, partition cannot select by channel or correlation, or average channels. It CAN select by spectral window(s).

Arguments

Inputs	
outputms	The name of the resulting measurement set allowed: string Default:
field	Fields to include, by names or 0-based ids. (" => all) allowed: variant Default:
spw	Spectral windows (and :channels) to select allowed: variant Default: *
baseline	Antenna names or indices to select (" => all) allowed: variant Default:
timebin	Duration for averaging. Defaults to no averaging. allowed: variant Default: -1s
time	Only use data in the given time range, using the msselection syntax. allowed: string Default:
scan	Only use the scan numbers requested using the msselection syntax. allowed: variant Default:
uvrange	Limit data by uv distance using the msselection syntax. allowed: variant Default:
taql	For the TAQL experts, flexible data selection using the TAQL syntax allowed: string Default:
whichcol	'DATA', 'MODEL_DATA', 'CORRECTED_DATA', 'FLOAT_DATA', 'LAG_DATA', and/or 'all' allowed: string Default: DATA
tileshape	Tile shape of the disk data columns, most users should not need to touch this parameter [0] => normal tiling, [1] => fast mosaic style tile [4,15,351] => a tile shape of 4 pol 15 chan and 351 rows allowed: variant Default:
subarray	Limit data to specific (sub)array numbers. allowed: variant Default:
combine	Ignore changes in these columns (scan, and/or state) when time averaging. allowed: string Default:
intent	Only use the requested scan intents. allowed: string Default:
obs	Only use the requested observation IDs. allowed: string Default:

Returns
bool

Example

```
ms.open("multiwin.ms")
ms.partition('partition.ms', field=[0], spw=[1], whichcol='CORRECTED_DATA')

In this example we partition out data from the 1st field and 2nd spectral
window.  Only the CORRECTED_DATA data column will be copied, and it will be
written to the DATA column of partition.ms.


ms.open("multiwin.ms")
ms.partition('partition.ms', field=[0], spw=[0,1,2,3],
             whichcol='CORRECTED_DATA')

In this example we partition out calibrated data from the 1st field and four
spectral windows.


ms.open("WSRT.ms")
ms.partition('partition.ms', timebin='20s', whichcol='all', combine='scan')
ms.close()

This example averages a WSRT MS into 20s bins, selecting whichever of DATA,
MODEL_DATA, CORRECTED_DATA, or FLOAT_DATA, or LAG_DATA is present.  Normally
the bins would not cross scans, but in this MS the scan number goes up with
each integration, making it redundant enough with time that it would defeat
any time averaging.  Therefore combine parameter forces the SCAN column to be
ignored for setting the bins.
```

---

ms.iterinit.html

ms.iterinit - Function

1.3.1 Initialize for iteration over an ms

Description

Specify the columns to iterate over and the time interval to use for the TIME column iteration. The columns are specified by their MS column name. Note that the following columns are always added to the specified columns: array_id, field_id, data_desc_id and time. This is so that the iterator can keep track of the coordinates associated with the data (field direction, frequency etc.) If you want to sort on these columns last instead of first you need to include them in the columns specified. If you don't want to sort on these columns at all, you can set adddefaultsortcolumns to False. You need to call selectinit before calling this. See the example below.

Arguments

Inputs		
columns	Vector of column names (case sensitive).	
	allowed:	stringArray
	Default:	
interval	Time interval in seconds (greater than 0), to group together in iteration	
	allowed:	double
	Default:	0.0
maxrows	Max number of rows (greater than 0) to return in iteration	
	allowed:	int
	Default:	0
adddefaultsortcolumns	Add the default sort columns	
	allowed:	bool
	Default:	true

Returns
bool

Example

See the example for the iterend function.

---

ms.iterorigin.html

ms.iterorigin - Function

1.3.1 Set the iterator to the start of the data.

Description

Set or reset the iterator to the start of the currently specified iteration. You need to call this before attempting to retrieve data with getdata. You can set the iteration back to the start before you reach the end of the data. You need to call iterinit before calling this. See the example below.

Arguments

Returns
bool

Example

```
See the example for the iterend function.
```

---

ms.iternext.html

ms.iternext - Function

1.3.1 Advance the iterator to the next lot of data

Description

This sets the currently selected table (as accessed with getdata) to the next iteration. If there is no more data, the function returns False and the selection is reset to that before the iteration started. You need to call iterinit and iterorigin before calling this. See the example below.

Arguments

Returns
bool

Example

```
See the example for the iterend function.
```

---

ms.iterend.html

ms.iterend - Function

1.3.1 End the iteration and reset the selected table

Description

This sets the currently selected table (as accessed with getdata) to the table that was selected before iteration started. Use this to end the iteration prematurely. There is no need to call this if you continue iterating until iternext returns False.
See the example below.

Arguments

Returns
bool

Example

```
ms.open("3C273XC1.MS")
ms.selectinit(datadescid=0)
ms.iterinit(["ANTENNA1","ANTENNA2","TIME"],60.0)
ms.iterorigin()
rec=ms.getdata(["u","v","data"])
ms.iternext()
ms.iterend()
```

```
We open the MS, select an array and spectral window and then
specify an iteration over interferometer and time, with a 60s time
interval.  We then set the iterator to the start of the data and
get out some data. Finally we advance the iterator to the next lot
of data and then end the iteration.
```

ms.fillbuffer.html

ms.fillbuffer - Function

1.3.1 Fill the internal buffer with data and flags.

Description

Read the specified data item from the table, including its flags and keep the results in an internal buffer

Arguments

Inputs		
item	data derived item	
	allowed:	string
	Default:	
ifraxis	Create interferometer axis if True	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
ms.open("3C273XC1.MS")
ms.select({'antenna1':[3]})
ms.fillbuffer("PHASE",True)
```

```
We open the MS for reading, select a subset and then
read the DATA, FLAG and FLAG\_ROW column, extract the PHASE,
reorder the data to add an interferometer axis,
and keep the results around in an internal buffer.
```

ms.diffbuffer.html

ms.diffbuffer - Function

1.3.1 Differentiate or difference the internal buffer.

Description

Subtract the previous point from each data point in the buffer (for window equal 2), or subtract the average over a window (for window greater than 2) from each point. The window can be in the time / row direction or the frequency / channel direction. The input data can be float or complex but the output is always float. The function returns statistics over the buffer: median for each time and channel, the average absolute deviation from the median in time and channel direction and over all pixels.

Arguments

Inputs		
direction	choose between time or channel direction: TIME or CHANNEL	
	allowed:	string
	Default:	TIME
window	width of averaging window in timeslots or channels; integer greater than 0	
	allowed:	int
	Default:	1

Returns

record

Example

```
ms.open("3C273XC1.MS")
ms.select({'antenna1':[3]})
ms.fillbuffer("DATA")
ms.diffbuffer("TIME",15)
```

```
# {'aad': array([[ 0.58959275],
#        [ 0.20988081],
#        [ 0.15907532],
#        [ 0.58837521]]),
# 'median': array([[ 2.67179847],
#        [ 0.32471114],
#        [ 0.37952924],
#        [ 2.60897708]])}
```

```
We open the MS for reading, select a subset and then
read the DATA, FLAG and FLAG\_ROW column,
we then subtract the average over a 15 point time-window from each
data point.
```

---

ms.getbuffer.html

ms.getbuffer - Function

1.3.1 Return the internal buffer as a Record for access from the intrepreter.

Description

Returns the internal buffer with either 'raw' or differenced data, flags and buffer statistics (if a difference operation was performed).

Arguments

Returns
record

Example

```
ms.open("3C273XC1.MS")
ms.select({'antenna1':[3]})
ms.fillbuffer("PHASE")
rec=ms.getbuffer()
```

```
We open the MS for reading, select a subset and then
read the DATA, FLAG and FLAG\_ROW column, extract the PHASE
and then obtain the results in a record.
```

---

ms.clipbuffer.html

ms.clipbuffer - Function

1.3.1 (NON-FUNCTIONAL???) Clip the internal buffer with specified limits.

Description

This sets flags in the internal buffer based on the clip levels specified. You can flag times, channels and individual pixels based on their deviation from the median. The cliplevel is specified in units of the corresponding average absolute deviation (a robust version of rms).

Arguments

Inputs		
pixellevel	cliplevel for pixels (greater than 0)	
	allowed:	double
	Default:	0.0
timelevel	cliplevel for time slots (greater than 0)	
	allowed:	double
	Default:	0.0
channellevel	cliplevel for channels (greater than 0)	
	allowed:	double
	Default:	0.0

Returns

bool

Example

```
ms.open("3C273XC1.MS")
ms.select({'antenna1':[3]})
ms.fillbuffer("DATA")
stats=ms.diffbuffer("TIME",15)
ms.clipbuffer(6,5,5)
#2008-05-28 17:15:27  SEVERE  casa::ms::open
```

```
#  Exception Reported: RecordInterface: field medTmeF is unknown
#---------------------------------------------------------------------------
#type 'exceptions.StandardError'         Traceback (most recent call last)
#
#/home/aips2mgr/testing/ipython console in module()
#
#type 'exceptions.StandardError': RecordInterface: field medTmeF is unknown
```

```
We open the MS for reading, select a subset and
read the data into the buffer. We then remove the average over a 15
point time window and clip the resulting data at 6 times the average
absolute deviation from the median for individual pixels, and at 5
times this for channels and timeslots.
```

---

ms.asdmref.html

ms.asdmref - Function

1.3.1 Test if the MS was imported with option lazy=True in importasdm and optionally change the ASDM reference

Description

If the MS is imported from an ASDM with option lazy=True, the DATA column of the MS is virtual and directly reads the visibilities from the ASDM. A reference to the original ASDM is stored with the MS. If the ASDM needs to be moved to a different path, the reference to it in the MS needs to be updated. This can be achieved with ms.asdmref(). When called with an empty string (default), the method just reports the currently set ASDM path.
Return value is a string containing the new path if the path was successfully set or (in the case abspath was empty) the MS indeed contains a ASDM reference, i.e. was lazily imported.
If the ASDM does not contain an ASDM reference, the method returns an empty string. If abspath is not empty and there was an error setting the new reference, the method throws an exception.

Arguments

Inputs	
abspath	new absolute path of the ASDM to be referenced (empty string = report current setting) allowed: string Default:

Returns
string

Example

```
Set the path to the referenced ASDM to "/home/alma/myanalysis/uid___A12345_X678_X910":
   ms.open("uid___A12345_X678_X910.ms",False)
   ms.asdmref("/home/alma/myanalysis/uid___A12345_X678_X910")
```

```
   ms.close()

Test if the MS was imported with lazy=True and therefore references an ASDM:
   ms.open("uid___A12345_X678_X910.ms")
   myref = ms.asdmref()
   ms.close()
   if myref=="":
     print "This MS does not reference an ASDM."
   else:
     print "This MS references the ASDM ", myref
```

---

ms.setbufferflags.html

ms.setbufferflags - Function

1.3.1 Set the flags in the buffer

Description

Replace the flag and flag_row fields in the internal buffer with those in the input record. The input record can be e.g., a modified version of the record returned by getbuffer(). The other fields in the record are ignored.

Arguments

Inputs		
flags	record with flag and flag_row	
	allowed:	record
	Default:	

Returns

bool

Example

```
ms.open("3C273XC1.MS",False)
ms.select({'antenna1':[3]})
ms.fillbuffer("PHASE")
rec=ms.getbuffer()
rec['flag_row'][17]=True
ms.setbufferflags(rec)
```

```
We open the MS for reading, select a subset and
read the data. We get the data into casapy, flag timeslot 17 and put
the modified flags back into the buffer.
```

ms.writebufferflags.html

ms.writebufferflags - Function

1.3.1 Write the flags in the internal buffer back to the table.

Description

Takes the flag and flag_row field in the internal buffer and writes them back to the FLAG and FLAG_ROW column in the currently selected table.

Arguments

Returns
bool

Example

```
ms.open("3C273XC1.MS",False)
ms.select({'antenna1':[3]})
ms.fillbuffer("PHASE")
rec=ms.getbuffer()
rec['flag_row'][17]=True
ms.setbufferflags(rec)
ms.writebufferflags()
```

```
We open the MS for reading, select a subset and
read the data. We get the data into casapy, flag timeslot 17 and put
the modified flags back into the buffer. We then write the buffer
flags back to the table, causing the corresponding data to be
marked flagged on subsequent access.
```

ms.clearbuffer.html

ms.clearbuffer - Function

1.3.1 Clear the internal buffer.

Description

Clears the internal buffer, returning the memory. This can be used after the (final) clipping/flagging operations have been performed.

Arguments

Returns

bool

Example

```
ms.open("3C273XC1.MS",False)
ms.select({'antenna1':[3]})
ms.fillbuffer("PHASE")
rec=ms.getbuffer()
rec['flag_row'][17]=True
ms.setbufferflags(rec)
ms.writebufferflags()
ms.clearbuffer()
```

```
We open the MS for reading, select a subset and
read the data. We get the data into casapy, flag timeslot 17 and put
the modified flags back into the buffer. We then write the buffer
flags back to the table, causing the corresponding data to be
marked flagged on subsequent access. Finally we clear the internal
```

buffer. This step can be omitted if you are about to do another fillbuffer().

---

ms.continuumsub.html

ms.continuumsub - Function

1.3.1 Continuum fitting and subtraction in uv plane

Description

This function provides a means of continuum determination and subtraction by fitting a polynomial of desired order to a subset of channels in each time-averaged uv spectrum. The fit is used to model the continuum in all channels (not just those used in the fit), for subtraction, if desired. Use the `fitspw` parameter to limit the spectral windows processed and the range of channels used to estimate the continuum in each (avoid channels containing spectral lines). The default solution interval 'int' will result in per-integration continuum fits for each baseline. The `mode` parameter indicates how the continuum model (the result of the fit) should be used: `'subtract'` will store the continuum model in the MODEL_DATA column and subtract it from the CORRECTED_DATA column; `'replace'` will replace the CORRECTED_DATA column with the continuum model (useful if you want to image the continuum model result); and `'model'` will only store the continuum model in the MODEL_DATA column (the CORRECTED_DATA is unaffected).
It is important to start the ms tool with `nomodify=False` so that changes to the dataset will be allowed (see example below).
For now, the only way to recover the un-subtracted CORRECTED_DATA column is to use calibrater.correct() again.
Note that the MODEL_DATA and CORRECTED_DATA columns must be present for `continuumsub` to work correctly. The function will warn the user if they are not present, and abort. To add these scratch columns (for now), close the ms tool, then start a calibrater or an imager tool, which will add the scratch columns. Then restart the ms tool, and try `continuumsub` again.
Options for shifting known bright sources to the phase center and for editing based on the rms fit will be added in the near future.

Arguments

Inputs		
field	Select fields to fit	
	allowed:	any
	Default:	variant
fitspw	Spectral windows/channels to use for fitting the continuum; default all spectral windows in all channels	
	allowed:	any
	Default:	variant
spw	Select spectral windows and channels from which to subtract a continuum estimate; default: all channels in all spectral windows for which the continuum was estimated	
	allowed:	any
	Default:	variant
solint	Continuum fit timescale (units optional)	
	allowed:	any
	Default:	variant int
fitorder	Polynomial order for fit	
	allowed:	int
	Default:	0
mode	Desired use of fit model (see below)	
	allowed:	string
	Default:	subtract

Returns
bool

Example

```
ms.fromfits('ngc5921.ms','/aips++/data/demo/NGC5921.fits')
ms.close()
cb.open('ngc5921.ms')  # add MODEL_DATA, CORRECTED_DATA columns
cb.close()
ms.open('ngc5921.ms',nomodify=False);    # writable!
ms.continuumsub(field=2,fitspw='0:5~9;50~59',
                solint=0.0,fitorder=1,mode='sub')
ms.done()
```

This example will fit a linear continuum to channels 5-9 and 50-59 in spectral window 0 in each scan-averaged spectrum for field 2, and store the result in the MODEL_DATA column and subtract it from the CORRECTED_DATA column.

---

ms.done.html

ms.done - Function

1.3.1 Closes the ms `tool`

Description

You should call close() when you are finished using the ms `tool` to close the measurement set table and free any associated file locks. The measurement set is not deleted.

Arguments

Returns
bool

Example

```
ms.open("3C273XC1.MS")
...
ms.done()
```

---

ms.msseltoindex.html

ms.msseltoindex - Function

1.3.1 Returns ids of the selection used

Description

Utility function that will return the ids of the selection used.

Arguments

Inputs	
vis	Measurementset for which this selection applies allowed: string Default:
spw	Spectral Window Ids (0 relative) to select; -1 interpreted as all allowed: any Default: variant
field	Field Ids (0 relative) or Field names (msselection syntax and wilcards are used) to select allowed: any Default: variantvariant
baseline	Antenna Ids (0 relative) or Antenna names (msselection syntax and wilcards are used) to select allowed: any Default: variant
time	Limit data selected to be within a given time range. Syntax is the defined in the msselection link allowed: any Default: variant
scan	Limit data selected on scan numbers. Syntax is the defined in the msselection link allowed: any Default: variant
uvrange	Limit data selected on uv distance. Syntax is the defined in the msselection link allowed: any Default: variant
observation	Select data by observation ID(s). The syntax is the same as for scan numbers. allowed: any Default: variant
polarization	Select data by polarization(s). allowed: any Default: variant
taql	For the TAQL experts, flexible data selection using the TAQL syntax allowed: string Default:

Returns
record

Example

```
a= ms.msseltoindex(vis='3C273XC1.MS', field='3C*')
print a['field']
# [0]
print a
#{'antenna1': array([], dtype=int32),
# 'antenna2': array([], dtype=int32),
#  'channel': array([], shape=(0, 0), dtype=int32),
#  'field': array([0]),
#  'scan': array([], dtype=int32),
#  'spw': array([], dtype=int32),
#  'obsids': array([], dtype=int32)}
```

```
Field name '3C*', in this case 3C273, corresponds to field id 0.

N.B.: The return values of unspecified fields (like antenna* and spw in the
      above example) will be left empty - this does not mean that selection
      excludes all antennas!

      Some fields (like 'field') are checked against the subtables of vis,
      but others are not.  For example, field='123~132' will produce an error
      if vis does not have fields 123 to 132, but for scan and obsids
      '123~132' would just return an array of integers from 123 to 132
      regardless of whether vis has those scan or observation IDs.  (The
      difference comes from it being quicker to check a subtable than the
      main table.)
```

---

ms.hanningsmooth.html

ms.hanningsmooth - Function

1.3.1 Hanning smooth the frequency channels to remove Gibbs ringing.

Description

This function Hanning smooths the frequency channels with a weighted running average of smoothedData[i] = 0.25*correctedData[i-1] + 0.50*correctedData[i] + 0.25*correctedData[i-1]. The first and last channels are flagged. Inclusion of a flagged value in an average causes that averaged data value to be flagged.

Arguments

Inputs		
datacolumn	the name of the MS column into which to write the smoothed data	
	allowed:	string
	Default:	corrected

Returns

bool

Example

```
ms.open('ngc5921.ms',nomodify=False)
ms.hanningsmooth('data')
ms.close()
```

ms.uvsub.html

ms.uvsub - Function

1.3.1 Subtract model from the corrected visibility data.

Description

This function subtracts model visibility data from corrected visibility data leaving the residuals in the corrected data column. If the parameter reverse is set True, this process is reversed.

Arguments

Inputs		
reverse	When False subtracts model from visibility data; when True adds model to visibility data	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
The following example subtracts a model from the visibility data leaving
the residuals in the corrected data column.

 ms.open('ngc5921.ms',nomodify=False)
 ms.uvsub()
 ms.close()

 The following example adds the model back into the residuals.

 ms.open('ngc5921.ms',nomodify=False)
 ms.uvsub(reverse=True)
 ms.close()
```

ms.addephemeris.html

ms.addephemeris - Function

1.3.1 Connect an ephemeris table with the MS FIELD table

Description

Arguments

Inputs	
id	The unique id number to give to this ephemeris (will overwrite pre-existing ephemeris of same id, -1 will use next unused id) allowed: int Default: -1
ephemerisname	The name of the ephemeris table which is to be copied into the MS allowed: string Default:
comment	Comment string (no spaces, will be part of a file name) allowed: string Default:
field	Field id(s) (0-based) or fieldname(s) to connect this ephemeris to allowed: any Default: variant

Returns
bool

Example

```
ms.addephemeris(id=0, ephemerisname="Titan_55002-55003dUTC.tab", comment="JPLTitan", f
```

ms.ngetdata.html

ms.ngetdata - Function

1.3.1 Read values from the measurement set. Use this method instead of the older getdata() method which is marked for deprecation.

Description

This method extracs the data as specified in the items parameter. The data is returned as a record with each item as a separate key in the record (all in lower case).
Unless the iterator was initialized with a niterinit(), this method initializes the iterator as niterinit([”..”],0.0,0,False).

Arguments

Inputs		
items	Item names (NOT USED)	
	allowed:	stringArray
	Default:	
ifraxis	Create interferometer axis if True (NOT USED)	
	allowed:	bool
	Default:	false
ifraxisgap	Gap size on ifr axis when antenna1 changes (NOT USED)	
	allowed:	int
	Default:	0
increment	Row increment for data access (NOT USED)	
	allowed:	int
	Default:	1
average	Average the data in time or over rows (NOT USED)	
	allowed:	bool
	Default:	false

Returns
record

ms.niterinit.html

ms.niterinit - Function

1.3.1 Initialize for iteration over an ms. Use this method instead of the older iterinit() method which is marked for deprecation.

Description

Arguments

Inputs	
columns	Vector of column names (case sensitive). This parameter is not used and is here only for backwards compatibility with the iterinit() method. allowed: stringArray Default:
interval	Time interval in seconds (greater than 0), to group together in iteration allowed: double Default: 0.0
maxrows	Max number of rows (greater than 0) to return in iteration. allowed: int Default: 0
adddefaultsortcolumns	Add the default sort columns allowed: bool Default: true

Returns

bool

---

ms.niterorigin.html

ms.niterorigin - Function

1.3.1 Set the iterator to the start of the data. Use this method instead of the older iterorigin() method which is marked for deprecation.

Description

Set or reset the iterator to the start of the currently specified iteration. You need to call this before attempting to iteratively retrieve data with ngetdata. You can set the iteration back to the start before you reach the end of the data. You need to call iterinit before calling this. See the example below.

Arguments

Returns
bool

Example

```
See the example for the niterend function.
```

---

ms.niternext.html

ms.niternext - Function

1.3.1 Advance the iterator to the next lot of data. Use this method instead of the older iternext() method which is marked for deprecation.

Description

This sets the currently selected table (as accessed with ngetdata) to the next iteration. If there is no more data, the function returns False. You need to call iterinit and iterorigin before calling this. See the example below.

Arguments

Returns
bool

Example

```
See the example for the niterend function.
```

---

ms.niterend.html

ms.niterend - Function

1.3.1 Query if there are more iterations left in the iterator. Use this method instead of the older iterend() method which is marked for deprecation.

Description

The serves redundant purpose and is here only for backward compatibility. This method returns True if there are no more iterations left. I.e., the iterations have ended. This same information is also returned by niternext(). With the use of the VisibilityIterator in the niterinit(), niterorigin(), niternext() methods, the iterator is set to the original state by calling niterinit() at any time.
See the example below.

Arguments

Returns
bool

Example

```
ms.open("3C273XC1.MS")
staql={'baseline':'1 & 2'};
ms.msselect(staql);
ms.niterinit([" "],60.0)
ms.niterorigin()
while (!ms.niterend()):
   rec=ms.ngetdata(["u","v","data"])
   ms.niternext()
ms.close()

We open the MS, select a baseline and then specify an iteration
```

```
over time, with a 60s time interval.  We then set the iterator
to the start of the data and get out some data.  We advance the
iterator to the next lot of data and continue till the end of
iterations is indicated.  Finally, we close the ms tool which
restores the tool to its original state.
```

---

msmetadata-Tool.html

1.3.2 msmetadata - Tool

Operations to retrieve metadata from a measurment set
Requires:

Synopsis

Description

The msmd tool provides methods to retrieve metadata from measurement sets.

Attaching to a Measurement Set
The simplest and most common way to attach an msmd tool to a measurement set is to use the msmd.open method which requires that you specify the name of the measurement set table.
NOTE: Any modifications to an MS while an associated msmd tool is open will not be reflected in the msmd tool. You must close and reopen the tool if you want to capture changes made to metadata of an MS if such a change occurs.

```
Example:

msmd.open("3C273XC1.MS")
# get the number of spectral windows
nspw = msmd.nspw()
msmd.done()
```

We open the tool by querying the MS for its metadata. We then get the number of spectral windows in the dataset and close the tool.

Methods

almaspws	Get a list of spectral window IDs with ALMA-specific attributes.
antennadiameter	Get the diameter for the specified antenna.
antennaids	Get the zero-based antenna ID for the specfied antenna name.
antennanames	Get the names of the antennas for the specfied zero-based antenna IDs.
antennaoffset	Get the offset position of the specified antenna relative to the array reference pos
antennaposition	Get the position of the specified antenna.
antennastations	Get the station names of the specified antennas.
antennasforscan	Get an array of the unique antenna IDs for the specified scan, obsservation ID, a
bandwidths	Get the bandwidths in Hz for the specified spectral windows. If spw less than ze
baseband	Get the baseband for the specified spectral window.

baselines	Get a two dimensional boolean array representing baselines for data recorded in
chanavgspws	Get an array of spectral window IDs used for channel averages. These are windo
chaneffbws	Get an array of channel effective bandwidths for the specified spectral window.
chanfreqs	Get an array of channel frequencies for the specified spectral window.
chanres	Get an array of channel resolutions for the specified spectral window.
chanwidths	Get an array of channel widths for the specified spectral window.
close	Close this tool and reclaim system resources associated with it.
corrprodsforpol	Get the correlation products associated with the specified polarization ID
corrtypesforpol	Get the correlation types associated with the specified polarization ID
datadescids	Get the data description IDs associated with the specified spectral window and/
done	Close this tool and reclaim system resources associated with it.
effexposuretime	Get the effective exposure (on-source integration time)
exposuretime	Get the exposure time for the specified scan, spwid, polarizaiton ID, array ID, ar
fdmspws	Get an array of spectral window IDs used for FDM. These are windows that do
fieldnames	Get an array of field names as they appear in the FIELD table.
fieldsforintent	Get an array of the unique fields for the specified intent.
fieldsforname	Get an array of the unique, zero-based field IDs for the specified field name.
fieldsforscan	Get an array of the unique fields for the specified scan number, observation ID, a
fieldsforscans	Get an array of the unique fields for the specified scan numbers, observationID, a
fieldsforsource	Get an array of the unique fields for the specified source ID.
fieldsforspw	Get an array of the unique fields for the specified spectral window.
fieldsfortimes	Get an array of the unique, zero-based, field IDs for the specified time range (tim
intents	Get an array of the unique intents associated with the MS.
intentsforfield	Get an array of the unique intents for the specified field.
intentsforscan	Get an array of the unique intents for the specified scan, obsservation ID, and ar
intentsforspw	Get an array of the unique intents for the specified spectral window ID.
meanfreq	Get the mean frequency for the specified spectral window.
name	Get the name of the attached MS.
nantennas	Get the number of antennas associated with the MS.
namesforfields	Get the name of the specified field.
namesforspws	Get the name of the specified spws.
nbaselines	Get the number of baselines represented in the main MS table.
nchan	Get the number of channels associated with the specified spectral window.
ncorrforpol	Get the number of correlations for the specified polarization ID.
nfields	Get the number of fields associated with the MS.
nobservations	Get the number of observations associated with the MS from the OBSERVATIO
nspw	Get the number of spectral windows associated with the MS.
nstates	Get the number of states (from the STATE table) associated with the MS.
nscans	Get the number of scans associated with the MS.
nsources	Get the number of unique values from the SOURCE_ID column in the SOURCE
nrows	Get the number of visibilities (from the main table) associated with the MS.
observers	Get an array observers as they are listed in the OBSERVATIONS table.
observatorynames	Get an array of MS telescope (observatory) names as they are listed in the OBSI
observatoryposition	Get the position of the specified telescope.
open	Attach the MS metadata tool to the specified MS
phasecenter	Get the phasecenter direction from a field ID and time if necessary

pointingdirection	Get the pointing direction for antennas at the specified row number in the main
polidfordatadesc	Get the polarization ID associated with the specified data description ID.
projects	Get an array projects as they are listed in the OBSERVATIONS table.
propermotions	Get the values of the PROPER_MOTION column from the SOURCE table.
refdir	Get the reference direction from a field ID and time if necessary
reffreq	Get the reference frequency of the specified spectral window.
restfreqs	Get the rest frequencies from the SOURCE table for the specified source and spe
scannumbers	Get an array of the unique scan numbers associated with the MS for the specifie
scansforfield	Get an array of the unique scan numbers associated with the specified field, obse
scansforintent	Get an array of the unique scan numbers associated with the specified intent, ob
scansforspw	Get an array of the unique scan numbers associated with the specified zero-based
scansforstate	Get an array of the unique scan numbers for the specified state, observation ID,
scansfortimes	Get an array of the unique scan numbers for the specified time range (time-tol to
schedule	Get the schedule information for the specified observation ID.
sideband	Get the sideband for the specified spectral window.
sourcedirs	Get the values of the DIRECTION column from the SOURCE table.
sourceidforfield	Get the source ID from the field table for the specified field ID.
sourceidsfromsourcetable	Get the values of the SOURCE_ID column from the SOURCE table.
sourcenames	Get the values of the SOURCE_NAME column from the SOURCE table.
spwsforbaseband	Get the spws associated with the specified baseband or dictionary that maps bas
spwfordatadesc	Get the spectral window ID associated with the specified data description ID.
spwsforfield	Get an array of the unique spectral window IDs for the specified field.
spwsforintent	Get an array of the unique spectral window IDs for the specified intent.
spwsforscan	Get an array of the unique spectral window IDs for the specified scan number, ob
statesforscan	Get an array of the unique state IDs for the specified scan number, observation I
summary	Get dictionary summarizing the MS.
tdmspws	Get an array of spectral window IDs used for TDM. These are windows that hav
timerangeforobs	Get the time range for the specified observation ID
timesforfield	Get an array of the unique times for the specified field.
timesforintent	Get an array of the unique times for the specified intent.
timesforscan	Get the unique times for the specified scan number, observation ID, and array ID
timesforscans	Get an array of the unique times for the specified scan numbers, observation ID,
transitions	Get the spectral transitions from the SOURCE table for the specified source and
wvrspws	Get an array of spectral window IDs used for WVR. These are windows that hav

msmetadata.almaspws.html

msmetadata.almaspws - Function

1.3.2 Get a list of spectral window IDs with ALMA-specific attributes.

Description

Get spectral window IDs based on ALMA-specific criteria. The inputs are or'ed together to form the returned list. If complement=True, then the complement of the selection is returned.

Arguments

Inputs		
chavg	Get channel average spectral windows?	
	allowed:	bool
	Default:	false
fdm	Get FDM spectral windows?	
	allowed:	bool
	Default:	false
sqld	Get square law (i.e. total power) detector spectral windows?	
	allowed:	bool
	Default:	false
tdm	Get TDM spectral windows?	
	allowed:	bool
	Default:	false
wvr	Get WVR spectral windows?	
	allowed:	bool
	Default:	false
complement	Return the complement of the selected set?	
	allowed:	bool
	Default:	false

Returns
intArray

Example

```
msmd.open("my.ms")
# get all square law detector spectral window IDs
msmd.almaspws(sqld=True)
# get all spectral window IDs other than those associated with square law detectors
msmd.almaspws(sqld=True, complement=True)
```

---

msmetadata.antennadiameter.html

msmetadata.antennadiameter - Function

1.3.2 Get the diameter for the specified antenna.

Description

Get the diameter for the specified antenna. The antenna can be specified either by its zero-based ID from the ANTENNA table or by its name in that table. The returned dictionary is a valid quantity. If a negative integer is provided for the antenna, then all atenna diameters will be returned in a dictionary that has keys that are the antenna IDs and values that are dictionaries, each being a valid quantity representing the diameter for that antenna ID.

Arguments

Inputs		
antenna	Zero-based antenna in the ANTENNA table, or antenna name. A negative integer will cause all antenna diameters to be returned. allowed: any Default: variant -1	

Returns

record

Example

```
msmd.open("my.ms")
# Get the diameter of the antenna named "VB2"
diameter = msmd.antennadiameter("VB2")
msmd.done()
```

msmetadata.antennaids.html

msmetadata.antennaids - Function

1.3.2 Get the zero-based antenna ID for the specfied antenna name.

Description

Get the zero-based antenna IDs for the specfied antenna names and the specified diameter range. An array of unique IDs in order of the specified names is returned. If no names and no diameter range is specified, all IDs are returned.

Arguments

Inputs	
name	Antenna names (string or string array) for which to get the corresponding IDs. Note that * matches any number of characters of all character classes. allowed: any Default: variant
mindiameter	Minimum antenna diameter, expressed as a quantity. allowed: any Default: variant 0m
maxdiameter	Maximum antenna diameter, expressed as a quantity. allowed: any Default: variant 1pc

Returns

intArray

Example

```
msmd.open("my.ms")
# get the zero-based antenna IDs for the antenna named "VB2"
antenna_id = msmd.antennaids("VB2")[0]
# get the zero-based antenna IDs for all antennas with diameters between 9m and 11m
```

```
antenna_ids = msmd.antennaids(mindiameter="9m", maxdiameter=qa.quantity("11m"))
msmd.done()
```

---

msmetadata.antennanames.html

msmetadata.antennanames - Function

1.3.2 Get the names of the antennas for the specfied zero-based antenna IDs.

Description

Get the name of the antenna for the specfied zero-based antenna ID. If antennaids is not specified, all antenna names are returned.

Arguments

Inputs		
antennaids	Zero-based antenna IDs (int or int array) for which to get the antenna names.	
	allowed:	any
	Default:	variant -1
		-1

Returns
stringArray

Example

```
msmd.open("my.ms")
# get the name associated with antenna ID 31
antenna_name = msmd.antennanames(31)[0]
msmd.done()
```

msmetadata.antennaoffset.html

msmetadata.antennaoffset - Function

1.3.2 Get the offset position of the specified antenna relative to the array reference position.

Description

Get the offset position of the specified antenna relative to the array reference position. Antenna may be specified as a zero-based integer (row number in the ANTENNA table) or a string representing a valid antenna name. The returned record contains the longitude, latitude, and elevation offsets as quantity records. The reported longitude and latitude offsets are measured along the surface of a sphere whose center is coincident with the center of the earth and whose surface contains the observatory reference position.

Arguments

Inputs		
which	Zero-based antenna in the ANTENNA table, or antenna name.	
	allowed:	any
	Default:	variant 0

Returns

record

Example

```
msmd.open("my.ms")
# get the offset of the (zero-based) 3rd antenna in the ANTENNA table
antennna_offset = msmd.antennaoffset(3)
# get the offset of antenna DV02
antennna_offset = msmd.antennaoffset('DV02')
msmd.done()
```

msmetadata.antennaposition.html

msmetadata.antennaposition - Function

1.3.2 Get the position of the specified antenna.

Description

Get the position of the specified antenna. The returned record represents a position measure, and can be used as such by the measures (me) tool.

Arguments

Inputs		
which	Zero-based antenna ID in the ANTENNA table or antenna name.	
	allowed:	any
	Default:	variant 0

Returns

record

Example

```
msmd.open("my.ms")
# get the position of the (zero-based) 3rd antenna in the ANTENNA table
antennna_position = msmd.antennaposition(3)
# get the position of the antenna named DV07
antennna_position = msmd.antennaposition("DV07")
msmd.done()
```

---

msmetadata.antennastations.html

msmetadata.antennastations - Function

1.3.2 Get the station names of the specified antennas.

Description

Get the station names of the specified antennas.

Arguments

Inputs		
which	Zero-based antenna ID(s) in the ANTENNA table or antenna name(s). Single numeric id less than zero retrieves all station names.	
	allowed:	any
	Default:	variant -1

Returns

stringarray

Example

```
msmd.open("my.ms")
# get all station names
stations = msmd.antennastations(-1)
# get the stations of the antennas named DV07 and DV01
stations = msmd.antennaposition(["DV07", "DV01"])
msmd.done()
```

---

msmetadata.antennasforscan.html

msmetadata.antennasforscan - Function

1.3.2 Get an array of the unique antenna IDs for the specified scan, obsservation ID, and array ID.

Description

Get an array of the unique antennaIDs for the specified scan, observation ID, and array ID.

Arguments

Inputs		
scan	Scan number for which to return the intents.	
	allowed:	int
	Default:	-1
obsid	Observation ID. If less than 0, all observation IDs are used.	
	allowed:	int
	Default:	-1
arrayid	Array ID. If less than 0, all array IDs are used.	
	allowed:	int
	Default:	-1

Returns

intArray

Example

```
msmd.open("my.ms")
# get the antennas associated with scan 4 (all observation IDs, all array IDs)
antennas = msmd.antennasforscan(4)
msmd.done()
```

msmetadata.bandwidths.html

msmetadata.bandwidths - Function

1.3.2 Get the bandwidths in Hz for the specified spectral windows. If spw less than zero, return bandwidths for all spectral windows.

Description

Get the bandwidths in Hz for the specified spectral windows. If spw less than zero, return bandwidths for all spectral windows.

Arguments

Inputs		
spw	Spectral window IDs, if integer less than zero, return bandwidths for all spectral windows.	
	allowed:	any
	Default:	variant -1

Returns

anyvariant

Example

```
msmd.open("my.ms")
# get bandwdith for spectral window 2.
baseband = msmd.bandwidth(2)
msmd.done()
```

---

msmetadata.baseband.html

msmetadata.baseband - Function

1.3.2 Get the baseband for the specified spectral window.

Description

Get the baseband for the specified spectral window.

Arguments

Inputs		
spw	Spectral window ID.	
	allowed:	int
	Default:	

Returns

int

Example

```
msmd.open("my.ms")
# get baseband for spectral window 2.
baseband = msmd.baseband(2)
msmd.done()
```

---

msmetadata.baselines.html

msmetadata.baselines - Function

1.3.2 Get a two dimensional boolean array representing baselines for data recorded in the MS.

Description

Get a two dimensional boolean array representing baselines for data recorded in the MS. A value of True means there is at least one row in the MS main table for that baseline, False means no rows for that baseline. Autocorrelation "baseline" information is also present via the values along the diagonal.

Arguments

Returns

anyvariant

Example

```
msmd.open("my.ms")
# get the baseline matrix for this data set
baselines = msmd.baselines()
msmd.done()
```

---

msmetadata.chanavgspws.html

msmetadata.chanavgspws - Function

1.3.2 Get an array of spectral window IDs used for channel averages. These are windows that do have 1 channel.

Description

Get an array of spectral window IDs used for channel averages. These are windows that do have 1 channel.

Arguments

Returns
intArray

Example

```
msmd.open("my.ms")
# get the spectral window IDs used for channel averages.
chan_avg_spws = msmd.chanavgspws()
msmd.done()
```

---

msmetadata.chaneffbws.html

msmetadata.chaneffbws - Function

1.3.2 Get an array of channel effective bandwidths for the specified spectral window.

Description

Get an array of channel effective bandwidths for the specified spectral window. The parameter asvel indicates if velocity widths (True) or frequency widths (False) should be returned. The unit parameter specifies the units that the returned values should have. If empty (default), "Hz" will be used if asvel=False, or "km/s" will be used if asvel=True.

Arguments

Inputs		
spw	Spectral window ID.	
	allowed:	int
	Default:	
unit	Desired unit of returned quantities. Empty means "Hz" if asvel=False, "km/s" if asvel=True.	
	allowed:	string
	Default:	
asvel	Should return values be equivalent velocity widths?	
	allowed:	bool
	Default:	false

Returns

doubleArray

Example

```
msmd.open("my.ms")
# get the channel effective bandwidths for spectral window 2, in m/s
chan_ebw = msmd.chaneffbws(2, "m/s", True)
msmd.done()
```

msmetadata.chanfreqs.html

msmetadata.chanfreqs - Function

1.3.2 Get an array of channel frequencies for the specified spectral window.

Description

Get an array of channel frequencies for the specified spectral window.

Arguments

Inputs		
spw	Spectral window ID.	
	allowed:	int
	Default:	
unit	Convert frequencies to this unit.	
	allowed:	string
	Default:	Hz

Returns

doubleArray

Example

```
msmd.open("my.ms")
# get the channel frequencies for spectral window 2.
chan_freqs = msmd.chanfreqs(2)
msmd.done()
```

---

msmetadata.chanres.html

msmetadata.chanres - Function

1.3.2 Get an array of channel resolutions for the specified spectral window.

Description

Get an array of channel resolutions for the specified spectral window. The parameter asvel indicates if velocity widths (True) or frequency widths (False) should be returned. The unit parameter specifies the units that the returned values should have. If empty (default), "Hz" will be used if asvel=False, or "km/s" will be used if asvel=True.

Arguments

Inputs		
spw	Spectral window ID.	
	allowed:	int
	Default:	
unit	Desired unit of returned quantities. Empty means "Hz" if asvel=False, "km/s" if asvel=True.	
	allowed:	string
	Default:	
asvel	Should return values be equivalent velocity resolutions?	
	allowed:	bool
	Default:	false

Returns

doubleArray

Example

```
msmd.open("my.ms")
# get the channel resolutions for spectral window 2, in m/s
chan_res = msmd.chanres(2, "m/s", True)
msmd.done()
```

msmetadata.chanwidths.html

msmetadata.chanwidths - Function

1.3.2 Get an array of channel widths for the specified spectral window.

Description

Get an array of channel widths for the specified spectral window.

Arguments

Inputs		
spw	Spectral window ID.	
	allowed:	int
	Default:	
unit	Convert frequencies to this unit.	
	allowed:	string
	Default:	Hz

Returns

doubleArray

Example

```
msmd.open("my.ms")
# get the channel widths for spectral window 2.
chan_freqs = msmd.chanwidths(2)
msmd.done()
```

---

msmetadata.close.html

msmetadata.close - Function

1.3.2 Close this tool and reclaim system resources associated with it.

Description

This method will close the tool and reclaim system resources it has been using. Returns true if successful.

Arguments

Returns
bool

Example

```
msmd.open("my.ms")
# do things with tool
# finish, close tool and free up resources.
msmd.close()
```

---

msmetadata.corrprodsforpol.html

msmetadata.corrprodsforpol - Function

1.3.2 Get the correlation products associated with the specified polarization ID

Description

Get the correlation products associated with the specified polarization ID.

Arguments

Inputs		
pol	Polarization ID. Must be nonnegative.	
	allowed:	int
	Default:	-1

Returns

anyvariant

Example

```
msmd.open("my.ms")
# get correlation products for polarization ID 3
corrprods = msmd.corrprodsforpol(3)
msmd.done()
```

---

msmetadata.corrtypesforpol.html

msmetadata.corrtypesforpol - Function

1.3.2 Get the correlation types associated with the specified polarization ID

Description

Get the correlation types associated with the specified polarization ID.

Arguments

Inputs		
pol	Polarization ID. Must be nonnegative.	
	allowed:	int
	Default:	-1

Returns

intArray

Example

```
msmd.open("my.ms")
# get correlation types for polarization ID 3
corrtypes = msmd.corrtypesforpol(3)
msmd.done()
```

---

msmetadata.datadescids.html

msmetadata.datadescids - Function

1.3.2 Get the data description IDs associated with the specified spectral window and/or polarization ID

Description

Get a list of data description IDs associated with the specified spectral window ID and/or polarization ID. Values of less than zero for either means all IDs should be used in the selection.

Arguments

Inputs		
spw	Spectral window ID. Less than zero implies any,	
	allowed:	int
	Default:	-1
pol	Polarization ID. Less than zero implies any.	
	allowed:	int
	Default:	-1

Returns

intArray

Example

```
msmd.open("my.ms")
# get all data description IDs associated with spw 2.
msmd.datadescids(spw=2)
# same as before but limit the IDs returned to those associated with
# polarization ID 3
msmd.datadescids(spw=2, pol=3)
msmd.done()
```

msmetadata.done.html

msmetadata.done - Function

1.3.2 Close this tool and reclaim system resources associated with it.

Description

This method will close the tool and reclaim system resources it has been using. Returns true if successful.

Arguments

Returns
bool

Example

```
msmd.open("my.ms")
# do things with tool
# finish, close tool and free up resources.
msmd.done()
```

---

msmetadata.effexposuretime.html

msmetadata.effexposuretime - Function

1.3.2 Get the effective exposure (on-source integration time)

Description

Get the effective exposure time (equivalent to what might be more commonly known as total integration time or total sample time) is calculated by summing over all rows in the main MS table, excluding autocorrelations or rows where FLAG_ROW is false, thusly:
sum[over i] (exposure[i]*sum[over j](UFBW[i, j])/ncorrelations[i])/ nmaxbaselines
where exposure[i] is the value of EXPOSURE for the ith row, the inner sum is performed over each correlation for that row, UFBW is the unflagged fractional bandwidth is determined by summing all the widths of the unflagged channels for that correlation and dividing by the total bandwidth of all spectral windows observed at the timestamp of row i, ncorrelations is the number of correlations determined by the number of rows in the FLAG matrix for MS row i, and nmaxbaselines is the maximum number of antenna pairs, nantennas*(nantennas-1)/2, where nantennas is the number of antennas in the ANTENNA table. This method returns a quantity (a dictionary having a numerical value and a string unit).

Arguments

Returns
record

Example

```
msmd.open("my.ms")
# get the effective exposure time.
exposure_time = msmd.effexposuretime()
msmd.done()
```

msmetadata.exposuretime.html

msmetadata.exposuretime - Function

1.3.2 Get the exposure time for the specified scan, spwid, polarizaiton ID, array ID, and observation ID.

Description

Get the exposure time for the specified scan, spwid, polarizaiton ID, array ID, and observation ID. This is the exposure time of the record with the lowest time stamp of the records associated with these parameters. Returns a quantity dictionary. If polid is not specified (or specified and negative) and there is only one polarization ID in for the specified combination of scan, spwid, obsID, and arrayID, then that polarization ID is used. If there are multiple polarization IDs for the combination of other parameters, a list of these is logged and an empty dictionary is returned.

Arguments

Inputs		
scan	Scan number.	
	allowed:	int
	Default:	0
spwid	Spectral window ID.	
	allowed:	int
	Default:	0
polid	Polarization ID.	
	allowed:	int
	Default:	-1
obsid	Observation ID.	
	allowed:	int
	Default:	0
arrayid	Array ID.	
	allowed:	int
	Default:	0

Returns
record

Example

```
msmd.open("my.ms")
# get the exposure time for scan 1, spwid 2, and polid 3
# for obsid=0 and arrayid=0
integration_time = msmd.getexposuretime(scan=1, spwid=2, polid=3)
msmd.done()
```

---

msmetadata.fdmspws.html

msmetadata.fdmspws - Function

1.3.2 Get an array of spectral window IDs used for FDM. These are windows that do not have 64, 128, or 256 channels.

Description

Get an array of spectral window IDs used for FDM. These are windows that do not have 64, 128, or 256 channels.

Arguments

Returns
intArray

Example

```
msmd.open("my.ms")
# get the spectral window IDs used for FDM.
fdm_spws = msmd.fdmspws()
msmd.done()
```

---

msmetadata.fieldnames.html

msmetadata.fieldnames - Function

1.3.2 Get an array of field names as they appear in the FIELD table.

Description

Get an array of field names as they appear in the FIELD table.

Arguments

Returns
stringArray

Example

```
msmd.open("my.ms")
# get list of field names in the ms
fieldnames = msmd.fieldnames()
msmd.done()
```

---

msmetadata.fieldsforintent.html

msmetadata.fieldsforintent - Function

1.3.2 Get an array of the unique fields for the specified intent.

Description

Get an array of the unique fields for the specified intent. Note that * matches any number of characters of all character classes.

Arguments

Inputs		
intent	Intent (case sensitive) for which to return the fields.	
	allowed:	string
	Default:	
asnames	If true, return the field names. If false, return the zero-based field IDs.	
	allowed:	bool
	Default:	false

Returns

any

Example

```
msmd.open("my.ms")
# get the field names for intent "observe target"
field_names = msmd.fieldsforintent("observe target", True, regex=False)
# get the field IDs for intent "observe target"
field_IDs = msmd.fieldsforintent("observe target", False, regex=False)
# get all field IDs for all intents which contain 'WVR'
field_IDs = msmd.fieldsforIntent("*WVR*")
msmd.done()
```

msmetadata.fieldsforname.html

msmetadata.fieldsforname - Function

1.3.2 Get an array of the unique, zero-based field IDs for the specified field name.

Description

Get an array of the unique, zero-based field IDs for the specified field name. If the field name is the empty string (the default), a list of all unique field IDs in the main table of the MS will be returned.

Arguments

Inputs		
name	Field name (case sensitive) for which to return the fields.	
	allowed:	string
	Default:	

Returns

intArray

Example

```
msmd.open("my.ms")
# get the field IDs for field name "Enceladus"
fields = msmd.fieldsforname("Enceladus")
msmd.done()
```

msmetadata.fieldsforscan.html

msmetadata.fieldsforscan - Function

1.3.2 Get an array of the unique fields for the specified scan number, observation ID, and array ID.

Description

Get an array of the unique fields for the specified scan number, observation ID, and array ID.

Arguments

Inputs	
scan	Scan number for which to return the fields. allowed: int Default: -1
asnames	If true, return the field names. If false, return the zero-based field IDs. allowed: bool Default: false
obsid	Observation ID. A negative value means use all observation IDs. allowed: int Default: -1
arrayid	Array ID. A negative value means use all array IDs. allowed: int Default: -1

Returns
any

Example

```
msmd.open("my.ms")
# get the field names for scan number 5 (for all array IDs and all observation IDs).
field_names = msmd.fieldsforscan(5, True)
```

```
# get the field IDs for scan number 5 (for all array IDs and all observation IDs)
field_IDs = msmd.fieldsforscan(5, False)
msmd.done()
```

---

msmetadata.fieldsforscans.html

msmetadata.fieldsforscans - Function

1.3.2 Get an array of the unique fields for the specified scan numbers, observationID, and array ID.

Description

Get an array of the unique fields for the specified scan numbers, observation ID, and array ID.

Arguments

Inputs	
scans	Scan numbers for which to return the fields. allowed: intArray Default:
asnames	If true, return the field names. If false, return the zero-based field IDs. allowed: bool Default: false
obsid	Observation ID. A negative value means use all observation IDs. allowed: int Default: -1
arrayid	Array ID. A negative value means use all array IDs. allowed: int Default: -1

Returns

any

Example

```
msmd.open("my.ms")
# get the field names for scan numbers 5 and 10 (all obsids, all arrayids)
field_names = msmd.fieldsforscan([5, 10], True)
```

```
# get the field IDs for scan numbers 5 and 10 (all obsids, all arrayids)
field_IDs = msmd.fieldsforscan([5, 10], False)
msmd.done()
```

---

msmetadata.fieldsforsource.html

msmetadata.fieldsforsource - Function

1.3.2 Get an array of the unique fields for the specified source ID.

Description

Get an array of the unique fields for the specified spectral window.

Arguments

Inputs		
source	Zero-based source ID for which to return the fields.	
	allowed:	int
	Default:	-1
asnames	If true, return the field names. If false, return the zero-based field IDs.	
	allowed:	bool
	Default:	false

Returns

any

Example

```
msmd.open("my.ms")
# get the field names for source ID 1
field_names = msmd.fieldsforsource(1, True)
# get the field IDs for source ID 1
field_IDs = msmd.fieldsforsource(1, False)
msmd.done()
```

msmetadata.fieldsforspw.html

msmetadata.fieldsforspw - Function

1.3.2 Get an array of the unique fields for the specified spectral window.

Description

Get an array of the unique fields for the specified spectral window.

Arguments

Inputs		
spw	Zero-based spectral window ID for which to return the fields.	
	allowed:	int
	Default:	-1
asnames	If true, return the field names. If false, return the zero-based field IDs.	
	allowed:	bool
	Default:	false

Returns

any

Example

```
msmd.open("my.ms")
# get the field names for spectral window 1
field_names = msmd.fieldsforspw(1, True)
# get the field IDs for spectral window 1
field_IDs = msmd.fieldsforspw(1, False)
msmd.done()
```

msmetadata.fieldsfortimes.html

msmetadata.fieldsfortimes - Function

1.3.2 Get an array of the unique, zero-based, field IDs for the specified time range (time-tol to time+tol).

Description

Get an array of the unique, zero-based, fieldIDs for the specified time range (time-tol to time+tol).

Arguments

Inputs		
time	Time at center of time range.	
	allowed:	double
	Default:	-1
tol	Time on either side of center for specifying range.	
	allowed:	double
	Default:	0

Returns

intArray

Example

```
msmd.open("my.ms")
# get the field IDs associated with the specified time range
fields = msmd.fieldsfortimes(4.8428293714e+09, 20)
msmd.done()
```

---

msmetadata.intents.html

msmetadata.intents - Function

1.3.2 Get an array of the unique intents associated with the MS.

Description

Get an array of the unique intents associated with the MS.

Arguments

Returns
stringArray

Example

```
msmd.open("my.ms")
# get the intents associated with the MS
intents = msmd.intents()
msmd.done()
```

---

msmetadata.intentsforfield.html

msmetadata.intentsforfield - Function

1.3.2 Get an array of the unique intents for the specified field.

Description

Get an array of the unique intents for the specified field.

Arguments

Inputs		
field	Field ID or name for which to return the intents.	
	allowed:	any
	Default:	variant -1

Returns

stringArray

Example

```
msmd.open("my.ms")
# get the intents associated with field 4
intents = msmd.intentsforfield(4)
# get intents for field "MOS"
intents2 = msmd.intentsforfield("MOS")
msmd.done()
```

---

msmetadata.intentsforscan.html

msmetadata.intentsforscan - Function

1.3.2 Get an array of the unique intents for the specified scan, obsservation ID, and array ID.

Description

Get an array of the unique intents for the specified scan, observation ID, and array ID.

Arguments

Inputs		
scan	Scan number for which to return the intents.	
	allowed:	int
	Default:	-1
obsid	Observation ID. A negative value means use all observation IDs.	
	allowed:	int
	Default:	-1
arrayid	Array ID. A negative value means use all array IDs.	
	allowed:	int
	Default:	-1

Returns

stringArray

Example

```
msmd.open("my.ms")
# get the intents associated with scan 4 (all obsids, all arrayids)
intents = msmd.intentsforscan(4)
msmd.done()
```

msmetadata.intentsforspw.html

msmetadata.intentsforspw - Function

1.3.2 Get an array of the unique intents for the specified spectral window ID.

Description

Get an array of the unique intents for the specified spectral window ID.

Arguments

Inputs		
spw	Spectral window ID (>=0) for which to return the intents.	
	allowed:	int
	Default:	-1

Returns

stringArray

Example

```
msmd.open("my.ms")
# get the intents associated with spectral window ID 3
intents = msmd.intentsforspw(3)
msmd.done()
```

msmetadata.meanfreq.html

msmetadata.meanfreq - Function

1.3.2 Get the mean frequency for the specified spectral window.

Description

Get the mean frequency for the specified spectral window.

Arguments

Inputs		
spw	Spectral window ID.	
	allowed:	int
	Default:	
unit	Convert frequencies to this unit.	
	allowed:	string
	Default:	Hz

Returns

double

Example

```
msmd.open("my.ms")
# get the mean frequency for spectral window 2.
mean_freq = msmd.meanfreq(2)
msmd.done()
```

---

msmetadata.name.html

msmetadata.name - Function

1.3.2 Get the name of the attached MS.

Description

Get the name of the attached MS.

Arguments

Returns
string

Example

```
msmd.open("my.ms")
# get its name
myname = msmd.name()
msmd.done()
```

---

msmetadata.nantennas.html

msmetadata.nantennas - Function

1.3.2 Get the number of antennas associated with the MS.

Description

Get the number of antennas associated with the MS.

Arguments

Returns
int

Example

```
msmd.open("my.ms")
number_of_antennas = msmd.nantennas()
msmd.done()
```

---

msmetadata.namesforfields.html

msmetadata.namesforfields - Function

1.3.2 Get the name of the specified field.

Description

Get the name of the specified field.

Arguments

Inputs	
fieldids	Zero-based field IDs for which to get the names (integer or interger array). Unspecified will return all field names. allowed: any Default: variant

Returns

stringArray

Example

```
msmd.open("my.ms")
# get the name for field 8 and 2.
field_names = msmd.namesforfields([8, 2])
# get all field names
all_field_nams = namesforfields()
msmd.done()
```

---

msmetadata.namesforspws.html

msmetadata.namesforspws - Function

1.3.2 Get the name of the specified spws.

Description

Get the name of the specified spw(s).

Arguments

Inputs		
spwids	Zero-based spw ID(s) for which to get the names (integer or interger array). Unspecified will return all spw names.	
	allowed:	any
	Default:	variant

Returns

stringArray

Example

```
msmd.open("my.ms")
# get the name for spws 8 and 2.
spw_names = msmd.namesforspws([8, 2])
# get all spw names
all_spw_names = namesforspws()
msmd.done()
```

---

msmetadata.nbaselines.html

msmetadata.nbaselines - Function

1.3.2 Get the number of baselines represented in the main MS table.

Description

Get the number of unique baselines (antenna pairs) represented in the main MS table. This can, in theory, be less than n*(n-1)/2 (n being the number of antennas in the ANTENNA table), if data for certain baselines are not included in the main MS table. Autocorrelation "baselines" are included in this count if ac=True.

Arguments

Inputs		
ac	Include auto-correlation "baselines"?	
	allowed:	bool
	Default:	false

Returns

int

Example

```
msmd.open("my.ms")
number_of_baselines = msmd.nbaselines()
number_of_baselines_including_ac = msmd.nbaselines(True)
msmd.done()
```

msmetadata.nchan.html

msmetadata.nchan - Function

1.3.2 Get the number of channels associated with the specified spectral window.

Description

Get the number of channels associated with the specified spectral window.

Arguments

Inputs		
spw	Zero-based spw ID for which to get the number of channels.	
	allowed:	int
	Default:	

Returns

int

Example

```
msmd.open("my.ms")
nchan = msmd.nchan(3)
msmd.done()
```

---

msmetadata.ncorrforpol.html

msmetadata.ncorrforpol - Function

1.3.2 Get the number of correlations for the specified polarization ID.

Description

Get the number of correlations for the specified polarization ID. If the specified polarization ID is negative, an array of numbers of correlations is returned. The indices of that array represent polarization IDs.

Arguments

Inputs		
polid	Zero-based polarization ID. A negative number will cause all the numbers of correlations to be returned.	
	allowed:	int
	Default:	-1

Returns

variant

Example

```
msmd.open("my.ms")
# get the number of correlations associated with polarization ID 4
polid = msmd.ncorrforpol(4)
# get the array of numbers of correlations from the POLARIZATION table
polids = msmd.ncorrforpol(-1)
msmd.done()
```

---

msmetadata.nfields.html

msmetadata.nfields - Function

1.3.2 Get the number of fields associated with the MS.

Description

Get the number of fields associated with the MS.

Arguments

Returns
int

Example

```
msmd.open("my.ms")
number_of_fields = msmd.nfields()
msmd.done()
```

---

msmetadata.nobservations.html

msmetadata.nobservations - Function

1.3.2 Get the number of observations associated with the MS from the OBSERVATIONS table.

Description

Get the number of observations associated with the MS from the OBSERVATIONS table.

Arguments

Returns
int

Example

```
msmd.open("my.ms")
number_of_obs_ids = msmd.nobservations()
msmd.done()
```

---

msmetadata.nspw.html

msmetadata.nspw - Function

1.3.2 Get the number of spectral windows associated with the MS.

Description

This method will return the number of spectral windows in the associated MS.

Arguments

Inputs		
includewvr	Include wvr spectral windows? If false, exclude wvr windows from count.	
	allowed:	bool
	Default:	true

Returns

int

Example

```
msmd.open("my.ms")
number_of_spectral_windows = msmd.nspw()
msmd.done()
```

---

msmetadata.nstates.html

msmetadata.nstates - Function

1.3.2 Get the number of states (from the STATE table) associated with the MS.

Description

This method will return the number of states (number of rows in the STATES table) in the associated MS.

Arguments

Returns
int

Example

```
msmd.open("my.ms")
number_of_states = msmd.nstates()
msmd.done()
```

---

msmetadata.nscans.html

msmetadata.nscans - Function

1.3.2 Get the number of scans associated with the MS.

Description

Get the number of scans associated with the MS.

Arguments

Returns
int

Example

```
msmd.open("my.ms")
number_of_scans = msmd.nscans()
msmd.done()
```

---

msmetadata.nsources.html

msmetadata.nsources - Function

1.3.2 Get the number of unique values from the SOURCE_ID column in the SOURCE table.

Description

Get the number of unique values from the SOURCE_ID column in the SOURCE table. The number of rows in the SOURCE table may be greater than this value.

Arguments

Returns
int

Example

```
msmd.open("my.ms")
number_of_unique_source_ids = msmd.nsources()
msmd.done()
```

---

msmetadata.nrows.html

msmetadata.nrows - Function

1.3.2 Get the number of visibilities (from the main table) associated with the MS.

Description

Get the number of visibilities (from the main table) associated with the MS.

Arguments

Inputs	
autoc	Include autocorrelation data? If False, only cross correlation rows will be summed. allowed: bool Default: true
flagged	Include flagged data? If False, only unflagged or patially flagged rows will be summed. allowed: bool Default: true

Returns

double

Example

```
msmd.open("my.ms")
# get the total number of rows
nrows = msmd.nrows()
# got the number of cross correlation rows
ncross = msmd.nrows(auto=False)
# get the number of unflagged rows
ngood = msmd.nrows(flagged=False)
# get the number of unflagged cross correlation rows
ncrossunflagged = msmd.nrows(auto=False, flagged=False)
msmd.done()
```

msmetadata.observers.html

msmetadata.observers - Function

1.3.2 Get an array observers as they are listed in the OBSERVATIONS table.

Description

Get an array of observers as they are listed in the OBSERVATIONS table.

Arguments

Returns
stringArray

Example

```
msmd.open("my.ms")
# get the observers
observers = msmd.observers()
msmd.done()
```

---

msmetadata.observatorynames.html

msmetadata.observatorynames - Function

1.3.2 Get an array of MS telescope (observatory) names as they are listed in the OBSERVATIONS table.

Description

Get an array of MS telescope (observatory) names as they are listed in the OBSERVATIONS table.

Arguments

Returns
stringArray

Example

```
msmd.open("my.ms")
# get the telescope names
telescope_names = msmd.telescopenames()
msmd.done()
```

---

msmetadata.observatoryposition.html

msmetadata.observatoryposition - Function

1.3.2 Get the position of the specified telescope.

Description

Get the position of the specified telescope.

Arguments

Inputs		
which	Zero-based telescope position in the OBSERVATIONS table (see msmd.telescopenames()).	
	allowed:	int
	Default:	0

Returns

record

Example

```
msmd.open("my.ms")
# get the position of the 0th telescope
telescope_position = msmd.telescopeposition(0)
msmd.done()
```

---

msmetadata.open.html

msmetadata.open - Function

1.3.2 Attach the MS metadata tool to the specified MS

Description

Attach this tool to the specified MS.

Arguments

Inputs		
msfile	Name of the existing measurement set	
	allowed:	string
	Default:	
maxcache	Maximum cache size, in megabytes, to use.	
	allowed:	float
	Default:	50

Returns
bool

Example

```
msmd.open("my.ms")
# do stuff and close it
msmd.done()
```

msmetadata.phasecenter.html

msmetadata.phasecenter - Function

1.3.2 Get the phasecenter direction from a field ID and time if necessary

Description

Get a direction measures for the phasecenter of the field id and time specified

Arguments

Inputs	
fieldid	Zero-based field ID for which to get the phasecenter; default fieldid=0 allowed: int Default: 0
epoch	Optional time, expressed as a measures epoch dictionary, if field id has a polynomial in time phasecenter or an ephemerides table attached to the ID. Default value means evaluate at the origin TIME in the FIELD table allowed: record Default:

Returns

record

Example

```
msmd.open("my.ms")
# get phasecenter for field ID 1
mydir = msmd.phasecenter(1);
# if the phasecenter is a polynomial or has an ephemerides attached to
# it a time is needed to get the phase direction
ep=me.epoch('utc', '2015/03/15/15:30:55')
mydir2=msmd.phasecenter(2, ep)
msmd.done()
```

msmetadata.pointingdirection.html

msmetadata.pointingdirection - Function

1.3.2 Get the pointing direction for antennas at the specified row number in the main MS table.

Description

Get the pointing direction for antennas at the specified row number in the main MS table. Returns a record containing the time, antenna IDs and corresponding pointing directions.

Arguments

Inputs	
rownum	Row number in the main MS table. allowed: int Default: 0
interpolate	Interpolate pointings in case the interval in the main table is shorter than that in the pointing table (often the case in fast-scanning in single dish observaitions) allowed: bool Default: false
initialrow	Initial guess of row index in pointing table to start search. allowed: int Default: 0

Returns

record

Example

```
msmd.open("my.ms")
# get the pointing directions for row ID 500
dirs = msmd.pointingdirection(500)
msmd.done()
```

msmetadata.polidfordatadesc.html

msmetadata.polidfordatadesc - Function

1.3.2 Get the polarization ID associated with the specified data description ID.

Description

Get the polarization ID associated with the specified data description ID. If the specified data description ID is negative, an array of polarization IDs is returned. The indices of that array represent data description IDs.

Arguments

Inputs		
ddid	Zero-based data description ID. A negative number will cause all the polarization IDs to be returned.	
	allowed:	int
	Default:	-1

Returns

variant

Example

```
msmd.open("my.ms")
# get the polarization ID associated with data description ID 3
polid = msmd.polidfordatadesc(3)
# get the array of polarization IDs in the order they appear in the DATA_DESCRIPTION tab
polids = msmd.polidfordatadesc(-1)
msmd.done()
```

msmetadata.projects.html

msmetadata.projects - Function

1.3.2 Get an array projects as they are listed in the OBSERVATIONS table.

Description

Get an array of projects as they are listed in the OBSERVATIONS table.

Arguments

Returns
stringArray

Example

```
msmd.open("my.ms")
# get the projects
projects = msmd.projects()
msmd.done()
```

---

msmetadata.propermotions.html

msmetadata.propermotions - Function

1.3.2 Get the values of the PROPER_MOTION column from the SOURCE table.

Description

Get the values of the DIRECTION column from the SOURCE table. Returns a dictionary in which the keys are the associated zero-based row numbers, represented as strings, in the SOURCE table. The associated values are two element dictionaries, with keys "longitude" and "latitude", containing the longitudinal and latidinal components of the proper motion, which are valid quantity dictionaries.

Arguments

Returns
record

Example

```
msmd.open("my.ms")
# get PROPER_MOTION column values from the SOURCE table
mu = msmd.propermotions()
msmd.done()
# the direction associated with zero-based row number 10
mu10 = mu["10"]
```

msmetadata.refdir.html

msmetadata.refdir - Function

1.3.2 Get the reference direction from a field ID and time if necessary

Description

Get a direction measure for the reference direction of the field and time specified

Arguments

Inputs	
field	Zero-based field ID or field name for which to get the reference direction; default field=0 allowed: any Default: variant 0
epoch	Optional time, expressed as a measures epoch dictionary, if associated field has a polynomial in time reference direction or an ephemerides table attached it. Default value means evaluate at the origin TIME in the FIELD table allowed: record Default:

Returns
record

Example

```
msmd.open("my.ms")
# get reference direction for field ID 1
mydir = msmd.refdir(1);
# if the reference direction is a polynomial or has an ephemerides attached to
# it a time is needed to get the reference direction
ep=me.epoch('utc', '2015/03/15/15:30:55')
mydir2=msmd.phasecenter(2, ep)
```

```
msmd.done()
```

---

msmetadata.reffreq.html

msmetadata.reffreq - Function

1.3.2 Get the reference frequency of the specified spectral window.

Description

Get the reference frequency of the specified spectral window. The returned frequency is in the form of a valid measures dictionary.

Arguments

Inputs		
spw	Zero-based spectral window ID.	
	allowed:	int
	Default:	-1

Returns

record

Example

```
msmd.open("my.ms")
# get the reference frequency for spw ID 20
reffreq = msmd.reffreq(20)
msmd.done()
```

---

msmetadata.restfreqs.html

msmetadata.restfreqs - Function

1.3.2 Get the rest frequencies from the SOURCE table for the specified source and spectral window.

Description

Get the rest frequencies from the SOURCE table for the specified source and spectral window. The return value will be a dictionary of frequency measures if the rest frequencies are defined for the specified inputs, or False if they do not.

Arguments

Inputs		
sourceid	Zero-based source ID (from the SOURCE::SOURCE_ID column).	
	allowed:	int
	Default:	0
spw	Zero-based spectral window ID.	
	allowed:	int
	Default:	0

Returns

anyvariant

Example

```
msmd.open("my.ms")
# get the rest frequencies for source ID 2 and spw ID 20
reffreq = msmd.restfreqs(2, 20)
msmd.done()
```

msmetadata.scannumbers.html

msmetadata.scannumbers - Function

1.3.2 Get an array of the unique scan numbers associated with the MS for the specified observation ID and array ID.

Description

This method will return an array of unique scan numbers in the associated MS for the specified observation ID and array ID.

Arguments

Inputs	
obsid	Observation ID. A negative value indicates all observation IDs should be used. allowed: int Default: -1
arrayid	Array ID. A negative value indicates all array IDs should be used. allowed: int Default: -1

Returns
intArray

Example

```
msmd.open("my.ms")
# scan numbers for all obsids and all arrayids
scan_numbers = msmd.scannumbers()
msmd.done()
```

msmetadata.scansforfield.html

msmetadata.scansforfield - Function

1.3.2 Get an array of the unique scan numbers associated with the specified field, observation ID, and array ID.

Description

Get an array of the unique scan numbers associated with the specified field, observation ID, and array ID.

Arguments

Inputs	
intent	Field ID or field name (case sensitive) for which to return the scan numbers. allowed: any Default: variant
obsid	Observation ID. A negative value indicates all observation IDs should be used. allowed: int Default: -1
arrayid	Array ID. A negative value indicates all array IDs should be used. allowed: int Default: -1

Returns
intArray

Example

```
msmd.open("my.ms")
# get the scan numbers associated with field "planet Z" (all obsids, all arrayids)
scan_numbers = msmd.scansforfield("planet Z")
# get the scan numbers associated with field ID 5 (all obsids, all arrayids)
```

```
scan_numbers = msmd.scansforfield(5)
msmd.done()
```

---

msmetadata.scansforintent.html

msmetadata.scansforintent - Function

1.3.2 Get an array of the unique scan numbers associated with the specified intent, observation ID, and arrayID.

Description

Get an array of the unique scan numbers associated with the specified intent, observation ID, and arrayID. The "*" character matches any number of characters from all character classes.

Arguments

Inputs	
intent	Intent (case-sensitive) for which to return the scan numbers. allowed: string Default:
obsid	Observation ID. A negative value indicates all observation IDs should be used. allowed: int Default: -1
arrayid	Array ID. A negative value indicates all array IDs should be used. allowed: int Default: -1

Returns

intArray

Example

```
msmd.open("my.ms")
# get the scan numbers associated with intent "detect planet X" (all obsids, all arrayid
scan_numbers = msmd.scansforintent("detect planet X", regex=False)
# got all the scan numbers associated with all intents which contain 'WVR' (all obsids,
```

```
scan_numbers = msmd.scansforintent("*WVR*")
msmd.done()
```

---

msmetadata.scansforspw.html

msmetadata.scansforspw - Function

1.3.2 Get an array of the unique scan numbers associated with the specified zero-based spectral window ID, observation ID, and array ID.

Description

Get an array of the unique scan numbers associated with the specified zero-based spectral window ID, observation ID, and array ID.

Arguments

Inputs	
spw	Zero-based spectral window ID for which to return the scan numbers. allowed: int Default: -1
obsid	Observation ID. A negative value indicates all observation IDs should be used. allowed: int Default: -1
arrayid	Array ID. A negative value indicates all array IDs should be used. allowed: int Default: -1

Returns

intArray

Example

```
msmd.open("my.ms")
# get the scan numbers associated with spectral window ID 14, all obsids, all arrayids
scan_numbers = msmd.scansforspw(14)
msmd.done()
```

msmetadata.scansforstate.html

msmetadata.scansforstate - Function

1.3.2 Get an array of the unique scan numbers for the specified state, observation ID, and array ID.

Description

Get an array of the unique scan numbers for the specified state, observation ID, and array ID.

Arguments

Inputs	
state	ID of state for which to return the scan numbers. allowed: int Default: -1
obsid	Observation ID. A negative value indicates all observation IDs should be used. allowed: int Default: -1
arrayid	Array ID. A negative value indicates all array IDs should be used. allowed: int Default: -1

Returns
intArray

Example

```
msmd.open("my.ms")
# get the scan numbers associated with state 2, all obsids, all arrayids
scans = msmd.scansforstate(2)
msmd.done()
```

msmetadata.scansfortimes.html

msmetadata.scansfortimes - Function

1.3.2 Get an array of the unique scan numbers for the specified time range (time-tol to time+tol), observation ID, and array ID.

Description

Get an array of the unique scan numbers for the specified time range (time-tol to time+tol), observation ID, and array ID.

Arguments

Inputs	
time	Time at center of time range. allowed: double Default: -1
tol	Time difference on either side of center for specifying range. allowed: double Default: 0
obsid	Observation ID. A negative value indicates all observation IDs should be used. allowed: int Default: -1
arrayid	Array ID. A negative value indicates all array IDs should be used. allowed: int Default: -1

Returns

intArray

Example

```
msmd.open("my.ms")
# get the scan numbers associated with the specified time range (all obsids, all array i
```

```
scans = msmd.scansfortimes(4.84282937e+09, 20)
msmd.done()
```

---

msmetadata.schedule.html

msmetadata.schedule - Function

1.3.2 Get the schedule information for the specified observation ID.

Description

Get the schedule information for the specified observation ID.

Arguments

Inputs		
obsid	Observation ID.	
	allowed:	int
	Default:	-1

Returns

stringArray

Example

```
msmd.open("my.ms")
# get the schdule information for observation ID = 2
schedule = msmd.schedule()[2]
msmd.done()
```

---

msmetadata.sideband.html

msmetadata.sideband - Function

1.3.2 Get the sideband for the specified spectral window.

Description

Get the sideband for the specified spectral window.

Arguments

Inputs		
spw	Spectral window ID.	
	allowed:	int
	Default:	

Returns

int

Example

```
msmd.open("my.ms")
# get sideband for spectral window 2.
sideband = msmd.sideband(2)
msmd.done()
```

---

msmetadata.sourcedirs.html

msmetadata.sourcedirs - Function

1.3.2 Get the values of the DIRECTION column from the SOURCE table.

Description

Get the values of the DIRECTION column from the SOURCE table. Returns a dictionary in which the keys are the associated row numbers, represented as strings, in the SOURCE table. Each value in the returned dictionary is a valid direction measure.

Arguments

Returns
record

Example

```
msmd.open("my.ms")
# get DIRECTION column values from the SOURCE table
sourcedirs = msmd.sourcedirs()
msmd.done()
# the direction associated with zero-based row number 10
dir10 = sourcedirs["10"]
# convert it to B1950, using the measure interface
dir10_B1950 = me.convert(dir10, "B1950")
```

---

msmetadata.sourceidforfield.html

msmetadata.sourceidforfield - Function

1.3.2 Get the source ID from the field table for the specified field ID.

Description

Get the source ID from the field table for the specified field ID.

Arguments

Inputs		
field	Zero-based field ID for which to return the source ID from the field table.	
	allowed:	int
	Default:	-1

Returns

int

Example

```
msmd.open("my.ms")
# get source ID associated with field ID 2
sourceid = msmd.sourceidforfield(2)
msmd.done()
```

msmetadata.sourceidsfromsourcetable.html

msmetadata.sourceidsfromsourcetable - Function

1.3.2 Get the values of the SOURCE_ID column from the SOURCE table.

Description

Get the values of the SOURCE_ID column from the SOURCE table. It is unfortunate that the SOURCE table has a column named SOURCE_ID, because implicitly the "ID" of a row in an MS subtable is generally meant to reflect a row number in that table, but that is not the case for the SOURCE table.

Arguments

Returns
intArray

Example

```
msmd.open("my.ms")
# get SOURCE_ID column values from the SOURCE table
sourceids = msmd.sourceidsfromsourcetable()
msmd.done()
```

---

msmetadata.sourcenames.html

msmetadata.sourcenames - Function

1.3.2 Get the values of the SOURCE_NAME column from the SOURCE table.

Description

Get the values of the SOURCE_NAME column from the SOURCE table.

Arguments

Returns
stringArray

Example

```
msmd.open("my.ms")
# get SOURCE_NAME column values from the SOURCE table
sourcenames = msmd.sourcenames()
msmd.done()
```

---

msmetadata.spwsforbaseband.html

msmetadata.spwsforbaseband - Function

1.3.2 Get the spws associated with the specified baseband or dictionary that maps baseband to spws.

Description

Get the spectral windows associated with the specified baseband or dictionary that maps baseband to spectral windows.

Arguments

Inputs	
baseband	Baseband number. If <0, return a dictionary mapping basebands to spws. allowed: int Default: -1
sqldmode	If "include", include SQLD windows, if "exclude", exclude SQLD windows, if "only", include only SQLD windows. Case insenstive, inimum match honored. allowed: string Default: include

Returns

variant

Example

```
msmd.open("my.ms")
# get the spectral window IDs associated with all the basebands in this dataset
basebandtospwdict = msmd.spwsforbasebands()
# get an array of spws associated with baseband 2.
spwsforbb2 = msmd.spwsforbasebands(2)
msmd.done()
```

msmetadata.spwfordatadesc.html

msmetadata.spwfordatadesc - Function

1.3.2 Get the spectral window ID associated with the specified data description ID.

Description

Get the spectral window ID associated with the specified data description ID. If the specified data description ID is negative, an array of spectral window IDs is returned. The indices of that array represent data description IDs.

Arguments

Inputs		
ddid	Zero-based data description ID. A negative number will cause all the spectral window IDs to be returned.	
	allowed:	int
	Default:	-1

Returns

variant

Example

```
msmd.open("my.ms")
# get the spectral window ID associated with data description ID 3
spw = msmd.spwfordatadesc(3)
# get the array of spectral window IDs in the order they appear in the DATA_DESCRIPTION
spws = msmd.spwfordatadesc(-1)
msmd.done()
```

msmetadata.spwsforfield.html

msmetadata.spwsforfield - Function

1.3.2 Get an array of the unique spectral window IDs for the specified field.

Description

Get an array of the unique spectral window IDs for the specified field.

Arguments

Inputs		
field	Field (case sensitive string or zero-based integer ID) for which to return the spectral window IDs.	
	allowed:	any
	Default:	variant

Returns

intArray

Example

```
msmd.open("my.ms")
# get the spectral window IDs associated with field "Fomalhaut"
spws = msmd.spwsforfield("Fomalhaut")
# get spectral window IDs associated with field ID 2
spws = msmd.spwsforfield(2)
msmd.done()
```

---

msmetadata.spwsforintent.html

msmetadata.spwsforintent - Function

1.3.2 Get an array of the unique spectral window IDs for the specified intent.

Description

Get an array of the unique spectral window IDs for the specified intent. The "*" character matches any number of characters from all character classes.

Arguments

Inputs		
intent	Intent (case sensitive) for which to return the spectral window IDs. allowed: string Default:	

Returns

intArray

Example

```
msmd.open("my.ms")
# get the spectral window IDs associated with "MY COOL INTENT"
spws = msmd.spwsforintent("MY COOL INTENT")
# got all the spw IDs associated with all intents which contain 'WVR'
scan_numbers = msmd.spwsforintent("*WVR*")
msmd.done()
msmd.done()
```

---

msmetadata.spwsforscan.html

msmetadata.spwsforscan - Function

1.3.2 Get an array of the unique spectral window IDs for the specified scan number, observation ID, and array ID.

Description

Get an array of the unique spectral window IDs for the specified scan number, observation ID, and array ID.

Arguments

Inputs	
scan	Scan number for which to return the spectral window IDs. allowed: int Default: -1
obsid	Observation ID. A negative value means that all observation IDs should be used. allowed: int Default: -1
arrayid	Array ID. A negative value means that all array IDs should be used. allowed: int Default: -1

Returns

intArray

Example

```
msmd.open("my.ms")
# get the spectral window IDs associated with scan number 20, all obsids, all arrayids.
spws = msmd.spwsforscan(20)
msmd.done()
```

msmetadata.statesforscan.html

msmetadata.statesforscan - Function

1.3.2 Get an array of the unique state IDs for the specified scan number, observation ID, and array ID.

Description

Get an array of the unique state IDs for the specified scan number, observation ID, and array ID.

Arguments

Inputs	
scan	Scan number for which to return the state IDs. allowed: int Default: -1
obsid	Observation ID. A negative value means that all observation IDs should be used. allowed: int Default: -1
arrayid	Array ID. A negative value means that all array IDs should be used. allowed: int Default: -1

Returns

intArray

Example

```
msmd.open("my.ms")
# get the state IDs associated with scan number 251, all obsids, all arrayids
states = msmd.statesforscan(251)
msmd.done()
```

msmetadata.summary.html

msmetadata.summary - Function

1.3.2 Get dictionary summarizing the MS.

Description

Get dictionary summarizing the MS.

Arguments

Returns
record

Example

```
msmd.open("my.ms")
# get the summary
summary = msmd.summary()
msmd.done()
```

---

msmetadata.tdmspws.html

msmetadata.tdmspws - Function

1.3.2 Get an array of spectral window IDs used for TDM. These are windows that have 64, 128, or 256 channels.

Description

Get an array of spectral window IDs used for TDM. These are windows that have 64, 128, or 256 channels.

Arguments

Returns
intArray

Example

```
msmd.open("my.ms")
# get the spectral window IDs used for TDM.
tdm_spws = msmd.tdmspws()
msmd.done()
```

---

msmetadata.timerangeforobs.html

msmetadata.timerangeforobs - Function

1.3.2 Get the time range for the specified observation ID

Description

Get the time range for the specified observation ID. The return value is a dictionary containing keys "begin" and "end". Each of the associated value are dictionaries representing epochs which are valid measure records. The values are taken directly from the OBSERVATION subtable; no half-intervals are added or subtracted.

Arguments

Inputs		
obsid	Zero-based observation ID for which to get the time range.	
	allowed:	int
	Default:	-1

Returns
record

Example

```
msmd.open("my.ms")
# get the time range associated with observation ID 3
timerange = msmd.timerangeforobs(3)
msmd.done()
```

msmetadata.timesforfield.html

msmetadata.timesforfield - Function

1.3.2 Get an array of the unique times for the specified field.

Description

Get an array of the unique times for the specified field.

Arguments

Inputs		
field	Zero-based field ID for which to return the times.	
	allowed:	int
	Default:	-1

Returns

doubleArray

Example

```
msmd.open("my.ms")
# get the times associated with field 3
times = msmd.timesforfields(3)
msmd.done()
```

---

msmetadata.timesforintent.html

msmetadata.timesforintent - Function

1.3.2 Get an array of the unique times for the specified intent.

Description

Get an array of the unique times for the specified intent.

Arguments

Inputs		
intent	Intent for which to return the times.	
	allowed:	string
	Default:	

Returns

doubleArray

Example

```
msmd.open("my.ms")
# get the times associated with intent "myintent"
times = msmd.timesforintent("myintent")
msmd.done()
```

---

msmetadata.timesforscan.html

msmetadata.timesforscan - Function

1.3.2 Get the unique times for the specified scan number, observation ID, and array ID.

Description

Get the unique times for the specified scan number, observation ID, and array ID. If perspw=True, the returned data structure is a dictionary that has keys representing zero-based spectral window IDs and values representing the unique values of the TIME column corrsponding to the specified scan and that corresponding spectral window ID. If False, an array of unique values from the TIME column for the specified scan is returned; there is no separation into spectral window IDs.

Arguments

Inputs	
scan	Scan number for which to return the times. allowed: int Default: -1
obsid	Observation ID. A negative value indicates all observation IDs should be used. allowed: int Default: -1
arrayid	Array ID. A negative value indicates all array IDs should be used. allowed: int Default: -1
perspw	Return output dictionary with keys representing spectral window IDs (True), or an array of all times (False). allowed: bool Default: false

Returns
anyvariant

Example

```
msmd.open("my.ms")
# get the times associated with scan number 10, all obsids, all arrayids.
times = msmd.timesforscans(10)
msmd.done()
```

---

msmetadata.timesforscans.html

msmetadata.timesforscans - Function

1.3.2 Get an array of the unique times for the specified scan numbers, observation ID, and array ID.

Description

Get an array of the unique times for the specified scan numbers, observation ID, and array ID.

Arguments

Inputs	
scans	Scan numbers for which to return the times. allowed: intArray Default: -1
obsid	Observation ID. A negative value indicates all observation IDs should be used. allowed: int Default: -1
arrayid	Array ID. A negative value indicates all array IDs should be used. allowed: int Default: -1

Returns

doubleArray

Example

```
msmd.open("my.ms")
# get the times associated with scan numbers 10 and 20, all obsids, all arrayids
times = msmd.timesforscans([10,20])
msmd.done()
```

msmetadata.transitions.html

msmetadata.transitions - Function

1.3.2 Get the spectral transitions from the SOURCE table for the specified source and spectral window.

Description

Get the spectral transitions from the SOURCE table for the specified source and spectral window. The return value will be an array of transitions if the transitions are defined for the specified inputs, or False if they do not.

Arguments

Inputs		
sourceid	Zero-based source ID (from the SOURCE::SOURCE_ID column).	
	allowed:	int
	Default:	0
spw	Zero-based spectral window ID.	
	allowed:	int
	Default:	0

Returns

anyvariant

Example

```
msmd.open("my.ms")
# get the transitions for source ID 2 and spw ID 20
reffreq = msmd.transitions(2, 20)
msmd.done()
```

msmetadata.wvrspws.html

msmetadata.wvrspws - Function

1.3.2 Get an array of spectral window IDs used for WVR. These are windows that have 4 channels.

Description

Get an array of spectral window IDs used for WVR. These are windows that have 4 channels. If complement is True, return the complement set instead (all non-wvr spw IDs).

Arguments

Inputs		
complement	If True, return all non-wvr spws.	
	allowed:	bool
	Default:	-false

Returns

intArray

Example

```
msmd.open("my.ms")
# get the spectral window IDs used for WVR.
wvr_spws = msmd.wvrspws()
msmd.done()
```

msplot-Tool.html

1.3.3 msplot - Tool

Plot data from a measurement set
Requires:

Synopsis

Description

The msplot `tool` is a plotting tool for a measurement sets.
The functionality of the msplot `tool` extends that of the tableplot `tool` to add knowledgable about measurement sets. The msplot `tool` does for measurement sets what the tableplot `tool` does for tables. Much of the functionality is similar to that of tableplot `tool` and it may be useful to read the tableplot `tool` documentation.
There is also a similar plotting tool for calibration data, calplot tool. The calplot `tool` documentation may be useful to read since the calplot `tool` is similar to msplot `tool`

Overview of msplot `tool` functionality
At present, the msplot `tool` plots from a **single** measurment sets only. Eventually data from more than one measurement set will be able to be accessed and plotted at the same time. Plots from the same MeasurementSet can be overlayed and more then one plotting panel can be created so different plots can be viewed simultaneously.

- **Opening and Closing -** Before doing any plots you must call mp.open with the measurment set to be plotted. When finished use mp.reset to reset the measurment set; clear any data seletion done with mp.setdata as well as resets all of the plotting options back their default values. mp.done to close the measurement set, and the MS plotter.
- **Plotting -** The msplot `tool` provides several common plots that can be called easily, but the mp.plotxy function is a generic function for plotting. The available common plots are: mp.array, mp.azimuth, mp.baseline, mp.elevation, mp.hourangle, mp.parallacticangle, mp.uvcoverage, mp.uvdist, mp.vischannel, and mp.vistime.

  There are a number of plotting options that can be set to change the color, labels, symbol, number of plotting panels, and many others. The mp.plotoptions function controls the plot options. It is important to note that once a plot option is set, it remains set until a subsequent call to the **plotoptions** method turns it off.

In addition iterative plots are provided. Iterative plots allow a user to iterate over some column plotting for that particular item. For example, a user may wish to plot the uv distances for each antenna separately by using the iteration value of 'ANTENNA' with the mp.dist function.

All of methods listed above have an interation parameter. Valid values for this parameter are: BASELINE, ANTENNA, FIELD, SPW, SCAN, FEED, and ARRAY_ID. To plot the the next plots in the iteration use the mp.iterplotnext. To stop the iterative plot use mp.iterplotstop. Note that a useful operation to perform before doing and interative plot is to set the subplot option to plot multiple plots with the mp.plotoptions. funcion. For example, subplot=131 will yeild a plot of three rows of plots.fntvwy

- **Data Sselection -** The mp.setdata function can be used to select which data is to be viewed, more precisely antennas, fields, UV ranges, and time selections can specified.

- **(Un)Flagging -** (Un)Flagging can be performed both in memory and on disk. Flagging has recently undergone some refactoring and it is now possible to save what has been flagged in stages (versions). This gives msplot the ability to undo flagging if so desired. Also is a new button on the plotting window that turns on the flagging mode, allowing users to interactively select the areas of interest. Soon there will be other buttons for (un)flaggig, and for displaying information about the flagged regions.

  To (un)flag data first make a plot, see the plotting section above. The next step is to mark a region(s) on the plot. This is done with the mp.markregion function. If a specific region is given to this function this region is used, otherwise the regions can be marked on the plotter by selecting square regions with the mouse.

  Now there are regions marked on the plot, there are three different actions that can be taken at this point. The mp.locatedata function displays information about each of the points in the marked regions. (**Warning:** if a lot of points are selected a large amount of data will be given to the logger, this can slow down the logger drastically!) The other two actions that can be taken are closely related mp.flagdata and mp.unflagdata, to flag or unflag data respectively. Both of these functions allow the flagging to be done in memory (default) or on disk.

  One final function for flagging is mp.clearflags. **Warning: This function clears all flags, all data will be unflagged with this method.**

Methods

msplot	Construct a msplot tool for plotting measurment sets
open	Set the measurement set to be plotted.

clearplot	Clear the plotting window or a particular panel, or all panels.
emperorsNewClose	Like the Emperor's New Clothes.
reset	Reset the state of MS plot back to its default state.
closeMS	Close the measurement set being used.
close	See done – close and done do the same thing.
done	Close the current MeasurementSet, and destroy the plotter – ending all plotting.
plotoptions	Set the style of the plot.
summary	List a short summary, description, of the data in the open measurment set.
setdata	Select a subset of the measurement set to operate on.
extendflag	Set the scope of flagging extension
avedata	Specify which data is to be averaged in the MS (or selected MS).
plot	
checkplot	
plotxy	A generic plotting routine for Measurement sets.
checkplotxy	Routine for checking the sanity of a plotxy plot.
iterplotstart	Plot the first set of iterative plots.
iterplotnext	Continue plotting on an iteration axes.
iterplotstop	Stop an iterative plot.
savefig	Save the currently plotted image.
markregion	Mark a rectangular region to flag or to investigate the data in the area.
flagdata	Set flags for all selected regions marked using mp.markregion()
unflagdata	Unset flags in all regions marked using mp.markregion() Similar to the mp.flagdata()
clearflags	Clear all flags in the table. Note: This clears *all* flags and should be used with cauti
locatedata	Print info about data selected using mp.markregion().
saveflagversion	Save current flags, applied to the current measurement set with a version name.
restoreflagversion	Restore flags for the current Measurement Set.
deleteflagversion	For the current measurement set delete a saved flag_version.
getflagversionlist	Print out a list of saved flag_versions, for the current Measurement Set.

msplot.msplot.html

msplot.msplot - Function

1.3.3 Construct a msplot tool for plotting measurment sets

Description

Create a msplot **tool** object.
This is the most commonly used constructor. It creates an msplot `tool` which is associated with a particular measurement set, and a tableplot tool that is used to do the plotting.

Arguments

Returns
msplotobject

---

msplot.open.html

msplot.open - Function

1.3.3 Set the measurement set to be plotted.

Description

Set the measurement set to be plotted. This method must be invoked before any of the other msplot `tool` functions.

Arguments

Inputs		
msname	measurement set name, including path	
	allowed:	string
	Default:	
dovel	whether to calculate velocity or not	
	allowed:	bool
	Default:	false
restfreq	a rest frequency quanta or transition name	
	allowed:	string
	Default:	
frame	frequency frame for spectral axis	
	allowed:	string
	Default:	
doppler	doppler mode	
	allowed:	string
	Default:	

Returns

bool

Example

```
# Open a msplot tool with a measurment set
  mp.open( msname='./data/3C273XC1.ms' );
```

msplot.clearplot.html

msplot.clearplot - Function

1.3.3 Clear the plotting window or a particular panel, or all panels.

Description

Clear the plotting window. Either clear the whole window (default) or a particular panel (specified by the subplot parameter).

Arguments

Inputs	
subplot	Three (or four) digits number: first digit for nrows, second for ncols, the rest for pannel number. allowed: int Default: 000

Returns

bool

Example

```
# open a MS dataset, Plot the array on the left and uvcoverage on the
# right.
# Also set the X and Y axes labels, and the title,
mp.open( msname='ngc5921.ms');
mp.plotoptions( subplot=121 )
mp.array()
mp.plotoptions( subplot=122 )
mp.uvcoverage()

# Now clear the uvcoverage plot area and plot the uvdist instead.
mp.clearplot( subplot=122 )
mp.uvdist()

# Now clear all plots
```

```
mp.clearplot()
```

---

msplot.emperorsNewClose.html

msplot.emperorsNewClose - Function

1.3.3 Like the Emperor's New Clothes.

Description

The mp.close() method has been a much contested method. With user's not really wanting to close things, but pretend to. So just to make all happy we have a much anticipated new close method, sure to bedazzle and shine!
Note that you may find the mp.reset function very useful for controlling the state of the msplot tool.

Arguments

Inputs

Returns

bool

---

msplot.reset.html

msplot.reset - Function

1.3.3 Reset the state of MS plot back to its default state.

Description

Reset the state of MS plot back to its default state. Calling this function will cause **all** of the plot options to be reset to their default values, and to reset any data selection performed by the mp.setdata().

Arguments

Inputs

Returns
bool

---

msplot.closeMS.html

msplot.closeMS - Function

1.3.3 Close the measurement set being used.

Description

Close the measurement set being used. As a a side affect any data selections (via mp.setdata or mp.sespectral) are reset to their initial state, and the plot options (set via mp.plotoptions) are also set to their initial stat.

Arguments

Inputs

Returns

bool

---

msplot.close.html

msplot.close - Function

1.3.3 See done – close and done do the same thing.

Description

See done

Arguments

Returns
bool

---

msplot.done.html

msplot.done - Function

1.3.3 Close the current MeasurementSet, and destroy the plotter – ending all plotting.

Description

End the msplot tool

Arguments

Returns
bool

---

msplot.plotoptions.html

msplot.plotoptions - Function

1.3.3 Set the style of the plot.

Description

Set the style of the plot. This function allows the plot title, axis labels, font, plotting color, plot symbol and many, other aspects of the plot to be set by the user, giving the user much flexibility over the look of their plots.
The various aspects that can be controlled by the user are as follows:

- **Labels:** The title that appears at the top of the plot is controlled by the paramete'r. The label along the x-axis and y-axis are set using the *xlabel* and *ylabel* parameters. If no values are not provided by the user then these labels are constructed from the data selected to be plotted.

  To control the font size of the labels use the *font* parameter. The x-axis and y-axis labels are always set to be 2pts. smaller then the title, which is set at the given font size specified.

- **Size Controls:** To control the size of the window, the *windowsize*, and *aspectratio* options can be set.

  To control the range of points plotted use the *plotrange* option. This plot option accepts either time strings in the form YYYY/MM/DD/hh:mm:ss or real values.

- **Data Point Sytles:** There are a number of plotoptions for controlling the color and style of the points plotted.

  The *plotsymbol* option set both the color and/or shape of the points plotted. It accepts the same syntax as that used by the **pylab** plot function. There are six different colors used: 'k' black, 'r'ed, 'g'reen, 'b'lue, 'c'yan, 'y'ellow, wnd 'w'hite. The plotsymbols include, but are not limited to '+', 'o', '-', and '–'. For a full list see the

  matplotlib documentation.

  The *markersize* and *linewidth* options control how big the plot symbols/lines are.

  The *multicolor* plot option is unique to CASA. It is used to specify whether or not different channels/correlations are plotted in different colors. The colors used can not be set by anyof the plot options. Basically, when plotting the colors are cycled through, changing whenever a different channel or correlation is encountered.

For large measurement sets it may be useful to plot only a portion of the measurment set. The *skipnrows* allows every nth row to be plotted rather then all of the plots.

- **Multiple plots:** One of the more useful abilities of the CASA plotters is the ability to plot several plot simultaneously, either side-by-side, and/or one on top of the other.

  The *subplot* option determines the number of panels to create, each panel contains a plot of some data. Although the *subplot* option is a single integer, it is really treated as three separate integers:

  - *nrows*: number of rows of panels,
  - *ncols*: number of columns of panels, and
  - *panel*: the panel number, which panel to plot on.

  For example, *subplot=132* specifies that there are three panels side-by-side (three columns of panels), and that we are plotting on the second panel. The top, left corner panel is panel number 1, and the panel number increases to the right first, continuing on the next row when the end of a row is reached. The example section shows some examples that create a number of different panel arrangements.

  Related to the multi-panel plots is the *removeoldpanels* option. This option when set to *True*, the default value, mimics the native matplot behviour, clearing any panels that lie partially or completely under a new panel being plotted. If it is set to *False* new panels could potentially plot overtop of old panels depending on the subplot values.

  In addtions to having plots there is the ability to overplot. The *overplot* option when set to true will, instead of clearing the panel that is currently being plotted on, plot over what is already there. When *overplot* is set to true the next plot is plotted over top of what is already there, and if the *plotsymbol* has not been specified the *msplot* tool will automatically pick a different color to plot with. An example where this may be useful if plotting different spectral windows separately, but on the same plot.

  Related to overplotting is the *replacetopplot* and *showflags* option. By default *replacetopplot* is set to *False*, but if it is set to *True* then only the last plot on a panel is replaced. This option comes in handy when a mistake has been made in the last plot. If the *showflags* option is set to *True* then the flagged data is plotted. A nice feature is the ability to plot the flagged and unflagged data on the same plot, by doing the same plot with both the *overplot* and *showflags* plot turned on. Note that flagged data is always plotted in a magenta color.

Arguments

Inputs	
subplot	Three (or four) digits number: first digit for nx, second for ny, the rest for pannel number. allowed: int Default: 111
plotsymbol	String specifying the colour to plot in, as well as the symbol to plot. This argument takes the same values as pylab plot command. Some of the valid symbols are: 'o', '+', ... allowed: string Default:
plotcolor	String specifying the colour to plot in. This can be one of the predefined pylab colour names. This over-rides the colour specified in plotsymbol. allowed: string Default:
multicolor	chan: means different channels in different colours, corr: means different correlations in different colours, both: means different correlations and channels in multicolour none: plot everything the same color. allowed: string Default: none
plotrange	Plot data within the specified range of values will be plotted. The range of values is given as a string in the form [xmin, xmax, ymin, ymax]. For most plots the xmin/max and ymin/max values are expected to be numeric values. However, for time plots, ie. where one or more of the axis is time, the xmin/max and ymin/max are expected to be strings in the from YYYY/MM/DD/hh:mm:ss. allowed: string Default:
timeplot	Indicate if the data is to be interpretted as time time values. Valid values are 'o'ff, 'x'-axis, 'y'-axis, 'b'oth axes. . allowed: char Default: 'o'
markersize	Specify the size (in pixels) of the markers being plotted. Markers are specified with the plotsymbol option. . allowed: double Default: 8.0
linewidth	Occasionally lines, rather then points, are plotted. This option allows the width of the plotted lines to be specified in points (pixels). allowed: double Default: 1.0
overplot	To do overplot or not. allowed: bool Default: false
replacetopplot	true : when overplot=false, replace the top-most layer only false : overplot=false always creates a fresh stack of plots. allowed: bool Default: false
removeoldpanels	true : mimic the native matplotlib behaviour of clearing up plots that lie partially or completely underneath

Returns
bool

Example

```
# open a MS dataset, set the plot options.
# Also set the X and Y axes labels, and the title,
  mp.open( msname='./data/3C273XC1.ms');
  labels := ['Amplitude vs UVdist','uvdist','amplitude'];
  mp.plotoptions( windowsize=5,aspectratio=0.8, fontsize=14.0, \
       xlabel='uvdist', ylabel='amplitude', title='Amplitude vs. UVdist',
       plotsymbol='g+' );

# Create 3 panels for plotting, and starting an iterative plot.  Three
# plots will be plotted for each iteration of the plot.
mp.open( './data/ngc5921.ms')
mp.plotoptions( subplot=311 )
mp.vischannel( column='data', what='amp', iteration='baseline' )

# Create 3 panels for plotting, but they are different sizes.
# Two panels at the top, smaller with the array and uvcoverage plots.
# A single wider panel at the bottom (the whole second row) containing
# the uvdistance plot.  The uvdistance plot, plots the corrected data
# overtop of the actual data.
mp.open( './data/ngc5921.ms');
mp.plotoptions( subplot=221 );
mp.array();
mp.plotoptions(subplot=222);
mp.uvcoverage();
mp.plotoptions( subplot=212 );
mp.uvdist();
mp.plotoptions( overplot=1, plotcolor=3);
mp.uvdist(column='corrected_data');

# Plot the flagged and unflagged data on the same plot, plotting the
# visibility amplitude vs. the channel.
mp.open(ngc5921PATH);
mp.vischannel();
mp.plotoptions( showflags=1, overplot=1 );
mp.vischannel();
```

msplot.summary.html

msplot.summary - Function

1.3.3 List a short summary, description, of the data in the open measurment set.

Description

List a summary, description, of the selected data in the open measurment set. The information that is displayed includes:
anntenna names field names scan numbers specral window list, including the number of channels for each one. correlations
Eventually the summary will include the time range and uv distance range as well.

Arguments

Inputs		
selected	Determine if we print a summary of the selected (true) data or a summary of the full measurement set (false).	
	allowed:	bool
	Default:	true

Returns

bool

Example

```
# create a msplot tool and set the subset data for plotting.
mp.open( msname='./data/3C273XC1.ms');

# View a summary of the whole measurement set.
mp.summary( selected=false);

# View what we've selected.
mp.setdata( "spw=3~7", antenna="0~200" );
mp.summary( selected=true );
```

msplot.setdata.html

msplot.setdata - Function

1.3.3 Select a subset of the measurement set to operate on.

Description

Select a subset of the measurement set. All plots will operate on this subset of the measurement set based on the values given.
All of the *Index* fields expect lists of integers. All of the *Name* fields accept strings. Where both indices and expressions are allowed as inputs (antenna's for example), they will be combined together when selecting the data.
The expression strings contain values separated by ','. The wildcard character '*' can be used with the names. The '¿' and '¡' characters can be used to indicate values that are greater then, or less then (respectively) a particular value.
Note: that integer values in the antennaNames list will be interpreted as indices. The '∼' indicates a range for values.
Spectral windows *names* are bit of a special case as channel information can be specified too. Spectral windows are in the folling format:
(spwlist):(channellist) where the spetral window and channel list follow the expression conventions listed above. "RR,LL,RL" "[RR LL RR]" "(XX)" where different types of polarizations are separated by a space or a comma.

Arguments

Inputs	
baseline	Baseline selection expression. allowed: string Default:
field	Field selection string. allowed: string Default:
scan	Scan selection string. allowed: string Default:
uvrange	UV Range selection string. allowed: string Default:
array	Array selection string. allowed: string Default:
feed	Feed selection string. allowed: string Default:
spw	Spectral Window selection string. allowed: string Default:
correlation	Correlation selection string. allowed: string Default:
time	Time selection values are in the form: YYYYMMMMDDhh:mm:ss∼YYYYMMDDhh:mm:ssŝtep where the first or second term may be dropped, and more then one range may be specified. Ranges are separted by a ",". The step value is a real value representing the number of seconds to skip or average, depending on the value given. allowed: string Default:

Returns
bool

Example

```
# create a msplot tool and set the subset data for plotting.
```

```
  mp.open( msname='./data/3C273XC1.ms');

# Select all antenna's that begin with VLA or N, field 1,2 and 3,
# 'RR' and 'LL' correlations, and spectral windows 3,4, and 5.
  mp.setdata( antennaNames='VLA:N*', fieldNames='1~3', correlations='RR, LL',\
      spwIndex=[3,4,5] );

# Select all fields, LR correlations, uvdists greater than 125 kilolambda,
# and times of June 27, 1989 at 3:31:40.
  mp.setdata( fieldNames'*', correlations='LR', uvDists='>125kl' \
      times='1989/06/27/03:31:40' );
```

---

msplot.extendflag.html

msplot.extendflag - Function

1.3.3 Set the scope of flagging extension

Description

Set the scope of flagging extension

Arguments

Inputs	
extendcorr	Indicate correlation based flagging extension. Valid values are: allhalf allowed: string Default:
extendchan	Indicate channel based flagging extension. Valid values are: all allowed: string Default:
extendspw	Indicate spectral window based flagging extension. Valid values are: all allowed: string Default:
extendant	Indicate antenna (baselines) based flagging extension. Valid values are: all allowed: string Default:
extendtime	Indicate time based flagging extension. Valid values are: all allowed: string Default:

Returns
bool

Example

```
# create a msplot tool and set the subset data for plotting.
  mp.open( msname='./data/3C273XC1.ms');

# TO COME
  mp.average( ??? )
```

---

msplot.avedata.html

msplot.avedata - Function

1.3.3 Specify which data is to be averaged in the MS (or selected MS).

Description

No description here at the moment. This method is under active development and is changing frequently. Documentation will be provided when development settles down.

Arguments

Inputs	
chanavemode	Indicate what if any averaging should be done on the selected channels.The channel selection and averaging is done via the spw parameter. Valid values are: none: No averaging, default value step: Plot every nth point scalarstep: incoherent average of every n points vectorstep: coherent average of every n points scalarchunk: incoherent average of blocks with n points vectorchunk: coherent average of blocks with n points allowed: string Default: none
corravemode	Indicate what if any averaging should be done on the selected correlations. See chanavemode for a detailed description of the valid values. Valid values are: none, step, scalarstep, vectorstep, scalarchunk, or vectorchunk allowed: string Default: none
datacolumn	Indicate the visibility data to be averaged. Valid values are: DATA CORRECTEDDATA MODELDATA allowed: string Default: DATA
averagemode	Indicate the mode for channel and/or time averaging. Valid values are: vector scalar allowed: string Default: vector
averagechan	Indicate the number of channels to average. The default value of 1 means no channel averaging. allowed: string Default: 1
averagetime	Indicate the length of time interval to average. Valid values are double values of time in seconds. The default value 0 means no time averaging. allowed: string Default: 0
averageflagged	Indicate either flagged or unflagged data to average. allowed: bool Default: false
averagescan	Indicate whether time averaging cross scan boundaries. allowed: bool Default: false
averagebl	Indicate whether averaging cross baseline boundaries. allowed: bool Default: false
averagearray	Indicate whether averaging cross array boundaries. allowed: bool Default: false
averagechanid	Indicate whether using averaged channel id or not. allowed: bool Default: false
averagevel	Indicate whether calculating averaged velocity id or not. allowed: bool Default: false

Returns
bool

Example

```
# create a msplot tool and set the subset data for plotting.
  mp.open( msname='./data/3C273XC1.ms');

# TO COME
  mp.average( ??? )
```

---

msplot.plot.html

msplot.plot - Function

1.3.3

Arguments

Inputs	
type	The type of plot to do. Valid values are strings for each of the msplot function, the valid strings are: "array" "azimuth", "baseline", "elevation", "hourangle", "parallacticangle", "uvcoverage", "uvdist", "vischannel", and "vistime" allowed: string Default:
column	Column name in main table of measurment set to plot. Valid values are: data, corrected, model, residual, and weight. allowed: string Default:
value	String: amp, phase allowed: string Default:
iteration	List of strings: Antenna1, Antenna2, Feed1, Feed2, Field_id, Scan_number, and Time. Spectral Window/Polarization_id(not available yet) allowed: stringArray Default:

Returns
bool

Example

```
# create a msplot tool.
mp.open( msname='./data/ngc7538.ms');

# Plot the antenna distribution.
mp.plot( type='array' );
```

```
# Plot the uv distances with the corrected data column plotted
# over the data column
mp.plotoptions( overplot=False, plotcolor='chocolate' );
mp.plot( type='uvdist', value='phase' );
mp.plotoptions( overplot=True, plotcolor='lemonchiffon' );
mp.plot( type='uvdist', column='corrected', value='phase' );

# Do an iterative plot on baselines, on the visibility amplitude/channel
# plot.  We display 6 plots at a time.
mp.clearplot();
mp.plotoptions( subplot=231 );
mp.plot( type='vischannel', column='data', value='amp', iteration='baseline' );
mp.iterplotnext();
mp.iterplotnext();
mp.iterplotstop();
```

---

msplot.checkplot.html

msplot.checkplot - Function

1.3.3

Arguments

Inputs	
plottype	The type of plot to check. Valid values are strings for each of the msplot function, the valid strings are: "array" "azimuth", "baseline", "elevation", "hourangle", "parallacticangle", "uvcoverage", "uvdist", "vischannel", "visfrequency", "vistime", and "visvelocity", allowed: string Default:
column	Column name in main table of measurment set to plot. Valid values are: data, corrected, model, residual, and weight. allowed: string Default: data
value	String: amp, phase allowed: string Default: amp
iteration	List of strings: Antenna1, Antenna2, Feed1, Feed2, Field_id, Scan_number, and Time. Spectral Window/Polarization_id(not available yet) allowed: stringArray Default:

Returns
bool

Example

```
# create a msplot tool.
mp.open( msname='./data/ngc7538.ms');

# select the data: spectral windows 0 and 1, channels 3 through 5
# and RL correlations
mp.setdata( spwNames=['(0,1):[3-5]'],  correlations=['RL'] );
```

```
mp.checkplot( plottype='uvdist', column='data', value='amp');
```

---

msplot.plotxy.html

msplot.plotxy - Function

1.3.3 A generic plotting routine for Measurement sets.

Description

Plot X versus Y for all meaningful columns in the MAIN table of a MS and derived quantities.

1. X and Y may be one of the followings:
   Antenna1, Antenna2, Feed1, Feed2, Field_id,
   ifr_number(not available yet), Scan_number,
   Time, channel(not available yet), uvdistance,
   frequency(not available yet),u, v, w,
   weight, data, model, corrected, residual
   (derived quantities will be listed later).

2. iteration axis may be one of the followings:
   Antenna1, Antenna2, Feed1, Feed2, Field_id, Scan_number,
   Time, Spectral Window/Polarization_id(not available yet)

Arguments

Inputs	
x	X-axis, a column name in measurement ests' main table. Valid values are: antenna1, antenna2, aziumth, baseline, channel, corrected, data, elevation, feed1, feed2, field_id, frequency, hourangle, ifr_number, model, parallactic_angle, residual, scan_number, time, u, uvdist, v, velocity, w, and weight. allowed: string Default: uvdist
y	Y-axis, a column name in the measurment sets' main table. Valid values are the same as the X-axis values. allowed: string Default: data
xcolumn	The column name in the measurement sets' msin table. Valid values are: data, corrected, model, and residual. allowed: string Default: data
ycolumn	The column name in the measurement sets' msin table. Valid values are: data, corrected, model, and residual. allowed: string Default: data
xvalue	String, needed if X or Y is data quantity: amp, phase, real, imag allowed: string Default: amp
yvalue	String, needed if X or Y is data quantity: amp, phase, real, imag allowed: string Default: amp
iteration	List of strings: Antenna1, Antenna2, Feed1, Feed2, Field_id, Scan_number, and Time. Ex: iteration over baselines : ['antenna1','antenna2'] Spectral Window/Polarization_id(not available yet) allowed: stringArray Default:

Returns
bool

Example

```
# create a msplot tool.
mp.open( msname='./data/ngc7538.ms');

# select data. correlations are separated by a space:
#    correlations=['RR RL']
mp.setdata( spwNames=['(0,1):[3-5]'],  correlations=['RL'] );

# Do an iterative plot over baselines, plotting uvdist vs. data for each antenna.
mp.plotxy( X='uvdist', Y='', column='data', iteration=['antenna1, 'antenna2']);
```

---

msplot.checkplotxy.html

msplot.checkplotxy - Function

1.3.3 Routine for checking the sanity of a plotxy plot.

Description

Do a sanity checks of all the inputs that have been given. This includes checking the plot option values (mp.plotoptions()), data selection (mp.setdata()), and spectral selections (mp.setspectral()).
Also included in the sanity checks is a check to see how many points will be plotted. If there are millions of points to be plotted, but do not want to wait for a large plot try setting the skipnrows or averagenrows plot options.

Arguments

Inputs	
x	X-axis, a column name in measurement ests' main table. Valid values are: antenna1, antenna2, aziumth, baseline, channel, corrected, data, elevation, feed1, feed2, field_id, frequency, hourangle, ifr_number, model, parallactic_angle, residual, scan_number, time, u, uvdist, v, velocity, w, and weight. allowed: string Default: uvdist
y	Y-axis, a column name in the measurment sets' main table. Valid values are the same as the X-axis values. allowed: string Default: data
xcolumn	The column name in the measurement sets' msin table. Valid values are: data, corrected, model, and residual. allowed: string Default: data
ycolumn	The column name in the measurement sets' msin table. Valid values are: data, corrected, model, and residual. allowed: string Default: data
xvalue	String, needed if X or Y is data quantity: amp, phase, real, and imag allowed: string Default: amp
yvalue	String, needed if X or Y is data quantity: amp, phase, real, and imag allowed: string Default: amp
iteration	List of strings: Antenna1, Antenna2, Feed1, Feed2, Field_id, Scan_number, and Time. Spectral Window/Polarization_id(not available yet) allowed: stringArray Default:

Returns
bool

Example

```
# create a msplot tool, do some data selections and
```

```
# plot option selection and  check the validty of them.
mp.open( msname='./data/3C273XC1.ms');
mp.setdata( antennaNames='VLA:N*', fieldNames='1~3', correlations='RR, LL',\
      spwIndex=[3,4,5] );
mp.plotoptions( windowsize=5,aspectratio=0.8, fontsize=14.0, \
       xlabel='uvdist', ylabel='amplitude', title='Amplitude vs. UVdist',
       plotsymbol='g+' );
mp.checkplotxy( X='uvdist', Y='', column='data', iteration=['antenna1, 'antenna2']);
```

---

msplot.iterplotstart.html

msplot.iterplotstart - Function

1.3.3 Plot the first set of iterative plots.

Description

Begin a series of plots using subtables constructed via an iteration axes, which is set in either the mp.plot or mp.plotxy methods. Use iterplotnext() to step through. Multi-panel plots as well as overplots are supported with this function. Overplots have a restriction in that both plots must have the same axies.
Only forward step through is allowed.

Arguments

Inputs

Returns
bool

Example

```
# create a msplot tool, select the data with field name 3C273 for plot,
# and initialize a plot of Amplitude vs UV distance for
# channel 1 and stokes 1, iterating over Antenna1, and creating
# two plot panels per iteration page.
  mp.open( msname=['./data/3C273XC1.ms']);
  plotoptions.nxpanels := 1;
  plotoptions.nypanels := 2;
  plotoptions.windowsize := 6;
  plotoptions.aspectratio := 1.2;
  plotoptions.fontsize := 14.0;
  mp.setdata( fieldNames=['3C273'] );
  labels := ['Amplitude vs UVdist (iterating over Antenna1)','uvdist','amplitude'];
  mp.plotxy( x='SQRT(SUMSQUARE(UVW[1:2]))' y='AMPLITUDE(DATA[1,1])', iteration='ANTENNA1' );
```

```
  mp.iterplotstart();
To iterate over baseline, for stokes 1, channel 1.
  plotopts.nxpanels := 1;
  plotopts.nypanels := 4;
  labels := ['Amplitude vs UVdist (iterating over Baseline)','uvdist','amplitude'];
  iteraxes := ['ANTENNA1','ANTENNA2'];
  mp.plotxy( x=['SQRT(SUMSQUARE(UVW[1:2]))' y='AMPLITUDE(DATA[1,1])'], iteration=iteraxes )
  mp.iterplotstart();
```

---

msplot.iterplotnext.html

msplot.iterplotnext - Function

1.3.3 Continue plotting on an iteration axes.

Description

Start/Continue plotting by stepping through the iteration axes.

Arguments

Returns
bool

Example

```
# Iterate through the data, plotting the uvdist for each antenna1
# The same plot can be achieved with mp.uvdist( iteration='antenna1' )
  mp.open( msname=['./data/3C273XC1.ms']);
  mp.plotoptions( subplot=121, windowsize=6, aspectratio=1.2, fontsize=14.0);
  mp.setdata( uvDists=['>25kl'] );
  mp.plotxy(X='uvdist',Y='',iteration='antenna1');
  mp.iterplotnext();
  mp.iterplotnext();
  mp.iterplotstop();
```

---

msplot.iterplotstop.html

msplot.iterplotstop - Function

1.3.3 Stop an iterative plot.

Description

To be called at the end of the plot iterations, or in between if desired.

Arguments

Inputs	
rmplotter	Indicates of the plot window should be removed (true)from the display or left (false) allowed: bool Default: false

Returns

bool

Example

```
# see the example for iterplotnext()
```

---

msplot.savefig.html

msplot.savefig - Function

1.3.3 Save the currently plotted image.

Description

Store the contents of the plot window in a file. The file format (type) is based on the file name, ie. the file extension given determines the format the file is saved as. The accepted formats are eps, ps, png, pdf, and svg.
Internally, this function uses the matplotlib pl.savefig function.
Note that if a full path is not given that the files will be saved in the current working directory.

Arguments

Inputs	
filename	Name the plot image is to be saved to. allowed: string Default:
dpi	Number of dots per inch (resolution) to save the image at. allowed: int Default: -1
orientation	Either landscape or portrait. Supported by the postscript format only. allowed: string Default:
papertype	Valid values are: letter, legal, exective, ledger, a0-a10 and b0-b10. This option is supported byt the postscript format only. allowed: string Default:
facecolor	Color of space between the plot and the edge of the square. Valid values are the same as those accepted by the plotcolor option. allowed: string Default:
edgecolor	Color of the outer edge. Valid values are the same as those accepted by the plotcolor option. allowed: string Default:

Returns
bool

Example

```
# Open a msplot tool with a measurment set
  mp.open( msname='./data/3C273XC1.ms' );
# Plot something and save it in a pdf file.
  mp.plot( 'uvdist' )
  mp.savefig( 'uvdist.pdf', edgecolor='black' )
```

---

msplot.markregion.html

msplot.markregion - Function

1.3.3 Mark a rectangular region to flag or to investigate the data in the area.

Description

Mark a rectangular region on the plot. Each call to `markflag` allows one region to be drawn. Any number of successive calls can be made. This function marks and stores a list of marked regions. These regions can then be (un)flagged, or information about the marked data can be retrieved. To flag the data the `(un)flagdata` function must be used and to find out information about the data the `locatedata` function must be used.
In the case of multi-panel plots, the subplot parameter must be specified with each call. The subplot value corresponds to a row-major ordering of panels, see the subplot plot option information.
Marking the region requires two consecutive mouse clicks at the two diagonally opposite corners. A hatched rectangle will appear over the selected region. Alternative a specific region can be given to this function with the `region` parameter.

Arguments

Inputs		
subplot	Three digits number: first digit for nx, second for ny, last for pannel number.	
	allowed:	int
	Default:	111
region	[xmin,ymin,xmax,ymax] bounding box	
	allowed:	doubleArray
	Default:	0.0

Returns
bool

Example

```
# mark 2 flag regions on a multi-panel plot, one in panel 1 and one
# in panel 2.
  tp.markflags(subplot=131, region=[100,-100,50,-50]);
  tp.markflags(subplot=221);
```

---

msplot.flagdata.html

msplot.flagdata - Function

1.3.3 Set flags for all selected regions marked using mp.markregion()

Description

Set flags for all regions marked using `markflags()`. The plot is automatically redrawn after applying flags.
If reduction TaQL functions such as `sum,mean` are used, flags corresponding to all accessed values will be modified. For example, with a measurement set table, flagging on the mean amplitude of stokes 1 and channels 1 to 5, given by 'MEAN(AMPLITUDE(DATA[1,1:5]))' results in flags being set for all 5 accessed channels.
For a measurement set, by default, flags are set only for accessed channels and stokes when the DATA column is used. However all channels/stokes can be flagged for the marked flag regions by setting the corresponding row flag.

Arguments

Inputs

Returns
bool

Example

```
# mark 2 flag regions on a multi-panel plot, one in panel 1 and one
# in panel 2. Then apply the flags and write to disk.
  mp.markflags(subplot=221, region=[0,15,10,30]);
  mp.markflags(subplot=222, region=[15,30,10,30]);
  mp.flagdata();
```

msplot.unflagdata.html

msplot.unflagdata - Function

1.3.3 Unset flags in all regions marked using mp.markregion() Similar to the mp.flagdata()

Description

Unset flags for all regions marked using `markflags()`. See the `flagdata()` function for more information.

Arguments

Inputs

Returns
bool

Example

```
# mark 2 flag regions on a multi-panel plot with three rows of plots.
# One region is marked on panel 1 and one region onpanel 2. Then the
# marked regions are applied unflagging data and writing the changes
# to disk.
  mp.markflags(subplot=311);
  mp.markflags(subplot=312);
  mp.unflagdata();
```

---

msplot.clearflags.html

msplot.clearflags - Function

1.3.3 Clear all flags in the table. Note: This clears *all* flags and should be used with caution.

Description

Currently, this function clears all flags from the table. This will be modified to allow for selective un-flagging of previously flagged regions (specified by indexing into a stored history of marked flag-regions).

Arguments

Returns

bool

Example

```
# clear all flags from the subset of the measurement set.
  mp := msplot( msname=['./data/3C273XC1.ms']);
  mp.setdata( spwIndex=[0] );
  mp.clearflags();
  mp.done();
```

---

msplot.locatedata.html

msplot.locatedata - Function

1.3.3 Print info about data selected using mp.markregion().

Description

New functionality that is being added to the plotting facilities, as a result we've purposely not put any description in as we are still exploring how this function should work.

Arguments

Inputs

Returns
bool

Example

---

msplot.saveflagversion.html

msplot.saveflagversion - Function

1.3.3 Save current flags, applied to the current measurement set with a version name.

Description

Arguments

Inputs	
versionname	Version name allowed: string Default:
comment	Comment for this flag table allowed: string Default:
merge	merge type: "replace" existing flag version, "and" logical AND with exisiting flag version, or "or" logical OR with existing flag version allowed: string Default: replace

Returns
bool

Example

msplot.restoreflagversion.html

msplot.restoreflagversion - Function

1.3.3 Restore flags for the current Measurement Set.

Description

Arguments

Inputs	
versionname	List of flag versions to restore from. allowed: stringArray Default:
merge	merge type: "replace" existing flag version, "and" logical AND with exisiting flag version, or "or" logical OR with existing flag version allowed: string Default: replace

Returns
bool

Example

msplot.deleteflagversion.html

msplot.deleteflagversion - Function

1.3.3 For the current measurement set delete a saved flag_version.

Description

Arguments

Inputs		
versionname	Version name	
	allowed:	stringArray
	Default:	

Returns
bool

Example

msplot.getflagversionlist.html

msplot.getflagversionlist - Function

1.3.3 Print out a list of saved flag_versions, for the current Measurement Set.

Description

Arguments

Inputs

Returns
bool

Example

---

measures-Module.html

1.4 measures - Module

Measures handling

Description A measure is a quantity with a specified reference frame (e.g. UTC, J2000, mars). The measures module provides an interface to the handling of measures. The basic functionality provided is:

- *Conversion* Conversion of measures, especially between different frames (e.g. UTC to LAST)

- *Calculation* Calculation of e.g. a rest frequency from a velocity and a frequency.

This functionality is provided in a command line interface.

Measures

Measures are e.g. an epoch, or coordinates, which have, in addition to values (as quantities), also a reference specification, and possibly an offset. They are represented as records with fields describing the various entities embodied in the measure. These entities can be obtained by the access methods *gettype*, *getref*, *getoffset*. *getvalue*.

Each measure has its own list of reference codes (see the individual methods for creating them, like *direction*). If an empty or no code reference code is given, the default code for that type of measure will be used (e.g. it is *J2000* for a direction). If an unknown code is given, this default is also returned, but with a warning message.

The values of a measure (like the right-ascension for a direction) are given as quantities. Each of them can be either a scalar quantity with a scalar or vector for its actual value (see the following example). E.g a vector of length 2 of quanta will be seen in a direction constructor as a longitude and a latitude.

```
"""
#
print "\t----\t Module Ex 1 \t----"
print me.epoch('utc','today')      # note that your value will be different
#{'type': 'epoch', 'm0': {'value': 54175.865923379628, 'unit': 'd'}, 'refer': 'UTC'}
print me.direction('j2000','5h20m','-30.2deg')
#{'type': 'direction', 'm1': {'value': -0.52708943410228748, 'unit': 'rad'}, 'm0': {'value':
a = me.direction('j2000','5h20m','-30.2deg')
print me.gettype(a)
#Direction
print me.getoffset(a)
#{}
print me.getref(a)
#J2000
print me.getvalue(a)
#{'m1': {'value': -0.52708943410228748, 'unit': 'rad'}, 'm0': {'value': 1.3962634015954634,
print me.getvalue(a)['m0']
#{'value': 1.3962634015954634, 'unit': 'rad'}
print me.getvalue(a)['m1']
#{'value': -0.52708943410228748, 'unit': 'rad'}
print 'Last example! Exiting ...'
exit()
#
"""
```

Known measures are:

- epoch: an instance in time (internally expressed as MJD or MGSD)
- direction: a direction towards an astronomical object (including planets, sun, moon)
- position: a position on Earth
- frequency: electromagnetic wave energy
- radialvelocity: radial velocity of astronomical object
- doppler: doppler shift (i.e. radial velocity in non-velocity units like 'Optical', 'Radio'.)
- baseline: interferometer baseline
- uvw: UVW coordinates
- earthmagnetic: Earth' magnetic field

In addition to the reference code (like J2000), a measure needs sometimes more information to be convertable to another reference code (e.g. a time and position to convert it to an azimuth/elevation). This additional information is called the reference *frame*, and can specify one or more of 'where am i', 'when is it', 'what direction", 'how fast'.

The frame values can be set by the doframe tool function.

measures-Tool.html

1.4.1 measures - Tool

measures tool
Requires:

Synopsis

Methods

dirshow	Show direction measure as a string.
show	Show a measure as a string
epoch	define an epoch measure
direction	define a direction measure
getvalue	get the value of a measure
gettype	get the type of a measure
getref	get the reference code of a measure
getoffset	get the offset of a measure
cometname	get the current comet name
comettype	get the current comet table type
cometdist	get the distance of the current comet in the current frame
cometangdiam	get the angular diameter of the current comet in the current frame
comettopo	get the current comet table coordinates
framecomet	set the current comet table
position	define a position measure
observatory	get position of an observatory
obslist	get a list of known observatories
linelist	get a list of known spectral lines
spectralline	get frequency of a spectral line
sourcelist	get a list of known sources
source	get direction of a source
frequency	define a frequency measure
doppler	define a doppler measure
radialvelocity	define a radialvelocity measure
shift	Shift a direction measure by an offset angle at a position angle.
uvw	define a uvw measure
touvw	calculate a uvw measure from a baseline
expand	expand n positions to n*(n-1)/2 baselines
earthmagnetic	define an earthmagnetic measure
baseline	define a baseline measure
asbaseline	define a baseline from a position measure
listcodes	get known reference code names (list indices do not necessarily correspond to enumeration
measure	convert a measure to another reference
doframe	save a measure as frame reference
framenow	set the active frame time at now

showframe	show the currently active frame reference
toradialvelocity	convert a doppler type value to a real radial velocity
tofrequency	convert a doppler type value to a frequency
todoppler	convert a frequency or radialvelocity measure to a doppler measure
torestfrequency	convert a frequency and doppler measure to a rest frequency
rise	get rise and set sidereal time
riseset	get rise and set times
posangle	get position angle of two directions
separation	get separation angle between two directions
addxvalue	get some additional measure information
type	type of tool
done	free resources used by tool.
ismeasure	Check if measure

measures.dirshow.html

measures.dirshow - Function

1.4.1
Show direction measure as a string.

Description

dirshow will convert a direction measure to a string

Arguments

Inputs		
v	a direction measure value to be converted to string	
	allowed:	record
	Default:	

Returns

string

Example

```
print "\t----\t dirshow Ex 1 \t----"
print me.dirshow(me.direction('venus'))
#[0, 90] deg  VENUS
```

---

measures.show.html

measures.show - Function

1.4.1 Show a measure as a string

Description

show will convert a measure to a string.
All measures are catered for (at this moment *direction, position, epoch, radialvelocity, frequency, doppler, baseline, uvw, earthmagnetic*).

Arguments

Inputs		
v	measure value to be converted to string	
	allowed:	record
	Default:	
refcode	add the reference code to output	
	allowed:	bool
	Default:	true

Returns
string

Example

```
print "\t----\t show Ex 1 \t----"
print me.show(me.frequency('lsrk', qa.constants('HI')))
#1.42041e+09 Hz LSRK
print me.show(me.frequency('lsrk', qa.constants('HI')), refcode=false)
#1.42041e+09 Hz
```

measures.epoch.html

measures.epoch - Function

1.4.1 define an epoch measure

Description

epoch defines an epoch measure from the CLI. It has to specify a reference code, an epoch quantity value (see introduction for the action on a scalar quantity with either a vector or scalar value),
and optionally it can specify an offset, which in itself has to be an epoch.
Allowable reference codes are: *UTC TAI LAST LMST GMST1 GAST UT1 UT2 TDT TCG TDB TCB*.
Note that additional ones may become available. Check in CASA with:

```
print "\t----\t epoch Ex 1 \t----"
print me.listcodes(me.epoch())
#{'normal': ['LAST', 'LMST', 'GMST1', 'GAST', 'UT1', 'UT2', 'UTC', 'TAI',
# 'TDT', 'TCG', 'TDB', 'TCB', 'IAT', 'GMST', 'TT', 'ET', 'UT'], 'extra': []}
#
```

See quantity for possible time formats.

Arguments

Inputs		
rf	reference code	
	allowed:	string
	Default:	UTC
v0	epoch value	
	allowed:	any
	Default:	variant
off	optional offset epoch measure	
	allowed:	record
	Default:	

Returns

record

Example

```
print "\t----\t epoch Ex 2 \t----"
print me.epoch('utc','today')
#{'m0': {'value': 54048.861237743055, 'unit': 'd'},
# 'refer': 'UTC',
# 'type': 'epoch'}
```

---

measures.direction.html

measures.direction - Function

1.4.1 define a direction measure

Description

direction defines a direction measure from the CLI. It has to specify a reference code, direction quantity values (see introduction for the action on a scalar quantity with either a vector or scalar value),
and optionally it can specify an offset, which in itself has to be a direction.
Allowable reference codes are: *J2000 JMEAN JTRUE APP B1950 BMEAN BTRUE GALACTIC HADEC AZEL SUPERGAL ECLIPTIC MECLIPTIC TECLIPTIC MERCURY VENUS MARS JUPITER SATURN URANUS NEPTUNE PLUTO MOON SUN COMET.*
Note that additional ones may become available. Check in CASA with:

```
print "\t----\t direction Ex 1 \t----"
print me.listcodes(me.direction())
#{'normal': ['J2000', 'JMEAN', 'JTRUE', 'APP', 'B1950', 'BMEAN',
#'BTRUE', 'GALACTIC', 'HADEC', 'AZEL', 'AZELSW', 'AZELNE', 'AZELGEO',
#'AZELSWGEO', 'AZELNEGEO', 'JNAT', 'ECLIPTIC', 'MECLIPTIC',
#'TECLIPTIC', 'SUPERGAL', 'ITRF', 'TOPO', 'ICRS'], 'extra': ['MERCURY',
#'VENUS', 'MARS', 'JUPITER', 'SATURN', 'URANUS', 'NEPTUNE', 'PLUTO',
#'SUN', 'MOON', 'COMET']}
```

The direction quantity values should be longitude(angle) and latitude(angle) (none needed for planets: the frame epoch defines coordinates). See quantity for possible angle formats.

Arguments

Inputs		
rf	reference code	
	allowed:	string
	Default:	J2000
v0	longitude	
	allowed:	any
	Default:	variant
v1	latitude	
	allowed:	any
	Default:	variant
off	optional offset direction measure	
	allowed:	record
	Default:	

Returns
record

Example

```
print "\t----\t direction Ex 2 \t----"
print me.direction('j2000','30deg','40deg')
#{'m0': {'value': 0.52359877559829882, 'unit': 'rad'},
# 'm1': {'value': 0.69813170079773168, 'unit': 'rad'},
# 'refer': 'J2000',
# 'type': 'direction'}
#
print me.direction('mars')
#{'m0': {'value': 0.0, 'unit': 'rad'},
# 'm1': {'value': 1.5707963267948966, 'unit': 'rad'},
# 'refer': 'MARS',
# 'type': 'direction'}
```

measures.getvalue.html

measures.getvalue - Function

1.4.1 get the value of a measure

Description

getvalue gets the actual implementation value of the measure.

Arguments

Inputs		
v	measure (array of measures)	
	allowed:	record
	Default:	

Returns

record

Example

```
print "\t----\t getvalue Ex 1 \t----"
b=me.direction('j2000','0deg','80deg')
print me.getvalue(b)
#{'m0': {'value': 0.0, 'unit': 'rad'},
# 'm1': {'value': 1.3962634015954634, 'unit': 'rad'}}
```

---

measures.gettype.html

measures.gettype - Function

1.4.1 get the type of a measure

Description

gettype gets the actual type of the measure.

Arguments

Inputs		
v	measure (array of measures)	
	allowed:	record
	Default:	

Returns

string

Example

```
print "\t----\t gettype Ex 1 \t----"
b=me.direction('j2000','0deg','80deg')
print me.getvalue(b)
#{'m0': {'value': 0.0, 'unit': 'rad'},
# 'm1': {'value': 1.3962634015954634, 'unit': 'rad'}}
print me.gettype(b)
#'Direction'
```

---

measures.getref.html

measures.getref - Function

1.4.1 get the reference code of a measure

Description

gettype gets the actual reference code of the measure.

Arguments

Inputs		
v	measure (array of measures)	
	allowed:	record
	Default:	

Returns

string

Example

```
print "\t----\t getref Ex 1 \t----"
b=me.direction('j2000','0deg','80deg')
print me.getvalue(b)
#{'m0': {'value': 0.0, 'unit': 'rad'},
# 'm1': {'value': 1.3962634015954634, 'unit': 'rad'}}
print me.gettype(b)
#'Direction'
print me.getref(b)
#'J2000'
```

measures.getoffset.html

measures.getoffset - Function

1.4.1 get the offset of a measure

Description

getoff gets the actual offset of the measure (as a measure) or F if no offset given.

Arguments

Inputs		
v	measure (array of measures)	
	allowed:	record
	Default:	

Returns

record

Example

```
print "\t----\t getoffset Ex 1 \t----"
b=me.direction('j2000','0deg','80deg')
print me.getvalue(b)
#{'m0': {'value': 0.0, 'unit': 'rad'},
# 'm1': {'value': 1.3962634015954634, 'unit': 'rad'}}
print me.gettype(b)
#'Direction'
print me.getref(b)
#'J2000'
print me.getoffset(b)
#{}
```

measures.cometname.html

measures.cometname - Function

1.4.1 get the current comet name

Description

cometname gets the name of the current comet (if any).

Arguments

Inputs

Returns
string

Example

```
print "\t----\t cometname Ex 1 \t----"
print me.cometname()
#Thu Nov 9 21:27:25 2006      WARN :
#Method cometname fails! No Comet table present
#''
```

---

measures.comettype.html

measures.comettype - Function

1.4.1 get the current comet table type

Description

comettype gets the comet table type (apparent or topocentric)

Arguments

Inputs

Returns
string

Example

```
print "\t----\t comettype Ex 1 \t----"
print me.comettype()
# 'none'
```

---

measures.cometdist.html

measures.cometdist - Function

1.4.1 get the distance of the current comet in the current frame

Description

cometdist returns the distance in AU of the current comet in the current frame, as a quantity. It will return -1 AU on failure!

Arguments

Inputs

Returns

record

Example

```
print "\t----\t cometdist Ex 1 \t----"
# Directory with several Solar System ephemerides for setjy.
cometdir = os.getenv("CASAPATH").split()[0] + "/data/ephemerides/JPL-Horizons/"
me.framecomet(cometdir + "Ganymede_55438-56292dUTC.tab")
#  Out[5]: True
me.doframe(me.epoch("utc", "2011/01/03/17:00:00"))
me.doframe(me.observatory("ALMA"))
gandist = me.cometdist()
print gandist
# {'value': 5.1241088343892631, 'unit': 'AU'}
```

measures.cometangdiam.html

measures.cometangdiam - Function

1.4.1 get the angular diameter of the current comet in the current frame

Description

cometdist returns the angular diameter (as seen from Earth) in AU of the current comet in the current frame, as a quantity. It will return -1 radians on failure!

Arguments

Inputs

Returns

record

Example

```
print "\t----\t cometangdiam Ex 1 \t----"
# Directory with several Solar System ephemerides for setjy.
cometdir = os.getenv("CASAPATH").split()[0] + "/data/ephemerides/JPL-Horizons/"
me.framecomet(cometdir + "Ganymede_55438-56292dUTC.tab")
#  Out[5]: True
me.doframe(me.epoch("utc", "2011/01/03/17:00:00"))
me.doframe(me.observatory("ALMA"))
gad = me.cometangdiam()
print gad
# {'unit': 'rad', 'value': 6.8679673431729014e-06}
```

measures.comettopo.html

measures.comettopo - Function

1.4.1 get the current comet table coordinates

Description

comettopo gets the comet table's topographic coordinates used.

Arguments

Inputs

Returns
record

Example

```
print "\t----\t comettopo Ex 1 \t----"
print me.comettopo()
#Thu Nov 9 21:45:40 2006      WARN :
#Method comettopo fails!  No Topocentric Comet table present
#{'value': [0.0], 'unit': ''}
```

---

measures.framecomet.html

measures.framecomet - Function

1.4.1 set the current comet table

Description

framecomet will put the specified comet table in the frame.

Arguments

Inputs		
v	name of a table	
	allowed:	string
	Default:	

Returns

bool

Example

```
print "\t----\t framecomet Ex 1 \t----"
print me.framecomet('VGEO')
#True
print me.showframe()
#'Frame: VENUS comet between MJD 50802.7 and 50803.1'
print me.cometname()
#'VENUS'
print me.comettype()
#'APP'
print me.doframe(me.epoch('et',qa.quantity('1997/12/20/17:30:0')))
#True
print me.measure(me.direction('comet'),'app')
#{'m0': {'value': -0.94936485919663083, 'unit': 'rad'},
# 'm1': {'value': -0.34710256485894436, 'unit': 'rad'},
# 'refer': 'APP',
```

```
# 'type': 'direction'}
```

---

measures.position.html

measures.position - Function

1.4.1 define a position measure

Description

position defines a position measure from the CLI. It has to specify a reference code, position quantity values (see introduction for the action on a scalar quantity with either a vector or scalar value),
and optionally it can specify an offset, which in itself has to be a position.
Allowable reference codes are: *WGS84 ITRF* (World Geodetic System and International Terrestrial Reference Frame).
Note that additional ones may become available. Check in CASA with:

```
print "\t----\t position Ex 1 \t----"
print me.listcodes(me.position())
#{'normal': ['ITRF', 'WGS84'], 'extra': []}
```

The position quantity values should be either longitude (angle), latitude(angle) and height(length); or x,y,z (length). See quantity for possible angle formats.

Arguments

Inputs		
rf	reference code	
	allowed:	string
	Default:	WGS84
v0	longitude or x	
	allowed:	any
	Default:	variant
v1	latitude or y	
	allowed:	any
	Default:	variant
v2	height or z	
	allowed:	any
	Default:	variant
off	optional offset position measure	
	allowed:	record
	Default:	

Returns
record

Example

```
print "\t----\t position Ex 2 \t----"
print me.position('wgs84','30deg','40deg','10m')
#{'m0': {'value': 0.52359877559829882, 'unit': 'rad'},
# 'm1': {'value': 0.6981317007977319, 'unit': 'rad'},
# 'm2': {'value': 9.9999999999999982, 'unit': 'm'},
# 'refer': 'WGS84',
# 'type': 'position'}
print me.observatory('ATCA')
#{'m0': {'value': 2.6101423190348916, 'unit': 'rad'},
# 'm1': {'value': -0.5261379196128062, 'unit': 'rad'},
# 'm2': {'value': 6372960.2577234386, 'unit': 'm'},
# 'refer': 'ITRF',
# 'type': 'position'}

###One can use a quantity-vectors  especially when dealing with multiple antenna positions

ants=me.position('itrf',qa.quantity([3828763.11,3828746.55, 3828727.43],'m'), qa.quantity([4
            qa.quantity([5064923.01, 5064923.01, 5064923.51],'m'))

print ants

#{'m0': {'unit': 'rad',
#        'value': array([ 0.11504897,  0.11508633,  0.1150838 ])},
# 'm1': {'unit': 'rad',
#        'value': array([ 0.92031276,  0.92031276,  0.92031535])},
# 'm2': {'unit': 'm',
#        'value': array([ 6364639.28758924,  6364639.27051283,  6364627.33064587])},
# 'refer': 'ITRF',
# 'type': 'position'}
```

measures.observatory.html

measures.observatory - Function

1.4.1 get position of an observatory

Description

observatory will give you the position of an observatory as given in the system. At the time of writing the following observatories are recognised (but check e.g. the position GUI for currently known ones, or the me.obslist() tool function): *'ALMA' 'ARECIBO' 'ATCA' 'BIMA' 'CLRO' 'DRAO' 'DWL' 'GB' 'GBT' 'GMRT' 'IRAM PDB' 'IRAM_PDB' 'JCMT' 'MOPRA' 'MOST' 'NRAO12M' 'NRAO_GBT' 'PKS' 'SAO SMA' 'SMA' 'VLA' 'VLBA' 'WSRT' 'ATF' 'ATA' 'CARMA' 'ACA' 'OSF' 'OVRO_MMA' 'EVLA' 'ASKAP' 'APEX' 'SMT' 'NRO' 'ASTE' 'LOFAR' 'MeerKAT' 'KAT-7' 'EVN' 'LWA1' 'PAPER_SA' 'PAPER_GB' 'e-MERLIN' 'MERLIN2' 'Effelsberg' 'MWA32T'* .

Arguments

Inputs		
name	observatory name - case insensitive	
	allowed:	string
	Default:	ALMA

Returns

record

Example

```
print "\t----\t observatory Ex 1 \t----"
print me.observatory('ATCA')
#{'m0': {'value': 2.6101423190348916, 'unit': 'rad'},
# 'm1': {'value': -0.5261379196128062, 'unit': 'rad'},
# 'm2': {'value': 6372960.2577234386, 'unit': 'm'},
# 'refer': 'ITRF',
# 'type': 'position'}
```

measures.obslist.html

measures.obslist - Function

1.4.1 get a list of known observatories

Description

obslist will give you an array of strings of the observatories known in the Observatories table.

Arguments

Returns
stringArray

Example

```
print "\t----\t obslist Ex 1 \t----"
print me.obslist()

#['ALMA' 'ARECIBO' 'ATCA' 'BIMA' 'CLRO' 'DRAO' 'DWL' 'GB' 'GBT' 'GMRT'
 #'IRAM PDB' 'IRAM_PDB' 'JCMT' 'MOPRA' 'MOST' 'NRAO12M' 'NRAO_GBT' 'PKS'
 #'SAO SMA' 'SMA' 'VLA' 'VLBA' 'WSRT' 'ATF' 'ATA' 'CARMA' 'ACA' 'OSF'
 #'OVRO_MMA' 'EVLA' 'ASKAP' 'APEX' 'SMT' 'NRO' 'ASTE' 'LOFAR' 'MeerKAT'
 #'KAT-7' 'EVN' 'LWA1' 'PAPER_SA' 'PAPER_GB' 'e-MERLIN' 'MERLIN2'
 #'Effelsberg' 'MWA32T']
```

measures.linelist.html

measures.linelist - Function

1.4.1 get a list of known spectral lines

Description

linelist will give you a string with a space separated list of spectral lines known in the Lines table.
A number of lines are available now, but tables with many lines are already online, and will be interfaced once a nomenclature can be defined for the tens of thousands of lines.

Arguments

Returns
string

Example

```
print "\t----\t linelist Ex 1 \t----"
print me.linelist()
#'C109A CI CII166A DI H107A H110A H138B H166A H240A H272A
# H2CO HE110A HE138B HI OH1612 OH1665 OH1667 OH1720'
```

---

measures.spectralline.html

measures.spectralline - Function

1.4.1 get frequency of a spectral line

Description

spectralline will give you the frequency of a spectral line. The known list can be obtained by me.linelist().

Arguments

Inputs		
name	name	
	allowed:	string
	Default:	HI

Returns

record

Example

```
print "\t----\t spectralline Ex 1 \t----"
print me.spectralline('HI')
#{'m0': {'value': 1420405751.786, 'unit': 'Hz'},
# 'refer': 'REST',
# 'type': 'frequency'}
```

---

measures.sourcelist.html

measures.sourcelist - Function

1.4.1 get a list of known sources

Description

sourcelist will give you a string with the space separated list of sources known in the Sources table.

Arguments

Returns
string

Example

```
print "\t----\t sourcelist Ex 1 \t----"
print me.sourcelist()[0:62]
#'0002-478 0003+380 0003-066 0007+106 0007+171 0008-264 0008-421'
#......
```

---

measures.source.html

measures.source - Function

1.4.1 get direction of a source

Description

source will give you the direction of a source. The known list can be obtained by me.sourcelist().

Arguments

Inputs		
name	name	
	allowed:	any
	Default:	variant 1934-638

Returns

record

Example

```
print "\t----\t source Ex 1 \t----"
print me.source()
print me.source('1934-638')
#  Out[19]:
#{'m0': {'value': -1.1370073467795063, 'unit': 'rad'},
# 'm1': {'value': -1.1119959323803881, 'unit': 'rad'},
# 'refer': 'ICRS',
# 'type': 'direction'}
```

measures.frequency.html

measures.frequency - Function

1.4.1 define a frequency measure

Description

frequency defines a frequency measure from the CLI. It has to specify a reference code, frequency quantity value (see introduction for the action on a scalar quantity with either a vector or scalar value),
and optionally it can specify an offset, which in itself has to be a frequency.
Allowable reference codes are: *REST LSRK LSRD BARY GEO TOPO GALACTO LGROUP CMB*.
Note that additional ones may become available. Check in CASA with:

```
print "\t----\t frequency Ex 1 \t----"
print me.listcodes(me.frequency())
#{'normal': ['REST', 'LSRK', 'LSRD', 'BARY', 'GEO', 'TOPO',
# 'GALACTO', 'LGROUP', 'CMB'], 'extra': []}
```

The frequency quantity values should be in one of the recognised units (examples all give same frequency):

- value with time units: a period (0.5s)
- value as frequency: 2Hz
- value in angular frequency: 720deg/s
- value as length: 149896km
- value as wave number: 4.19169e-8m-1
- value as energy (h.nu): 8.27134e-9ueV
- value as momentum: 4.42044e-42kg.m

Arguments

Inputs		
rf	reference code	
	allowed:	string
	Default:	LSRK
v0	frequency/wavelength/...	
	allowed:	any
	Default:	variant
off	optional offset frequency measure	
	allowed:	record
	Default:	

Returns
record

Example

```
print "\t----\t frequency Ex 2 \t----"
print me.frequency('lsrk','5GHz')
#{'m0': {'value': 5000000000.0, 'unit': 'Hz'},
# 'refer': 'LSRK',
# 'type': 'frequency'}
print me.frequency('lsrk','21cm')
#{'m0': {'value': 1427583133.3333333, 'unit': 'Hz'},
# 'refer': 'LSRK',
# 'type': 'frequency'}
```

measures.doppler.html

measures.doppler - Function

1.4.1 define a doppler measure

Description

doppler defines a doppler measure from the CLI. It has to specify a reference code, doppler quantity value (see introduction for the action on a scalar quantity with either a vector or scalar value),
and optionally it can specify an offset, which in itself has to be a doppler.
Allowable reference codes are: *RADIO Z RATIO BETA GAMMA OPTICAL TRUE RELATIVISTIC*.
Note that additional ones may become available. Check in CASA with:

```
print "\t----\t doppler Ex 1 \t----"
print me.listcodes(me.doppler())
#{'normal': ['RADIO', 'Z', 'RATIO', 'BETA', 'GAMMA', 'OPTICAL',
# 'TRUE', 'RELATIVISTIC'], 'extra': []}
```

The doppler quantity values should be either non-dimensioned to specify a ratio of the light velocity, or in velocity.

Arguments

Inputs		
rf	reference code	
	allowed:	string
	Default:	RADIO
v0	doppler ratio/velocity	
	allowed:	any
	Default:	variant
off	optional offset doppler measure	
	allowed:	record
	Default:	

Returns

record

Example

```
Examples both give same doppler:

print "\t----\t doppler Ex 2 \t----"
print me.doppler('radio','0.4')
#{'m0': {'value': 119916983.2, 'unit': 'm/s'},
# 'refer': 'RADIO',
# 'type': 'doppler'}
print me.doppler('radio',qa.mul(qa.quantity('0.4'),qa.constants('c')))
#{'m0': {'value': 119916983.2, 'unit': 'm/s'},
# 'refer': 'RADIO',
# 'type': 'doppler'}
```

---

measures.radialvelocity.html

measures.radialvelocity - Function

1.4.1 define a radialvelocity measure

Description

radialvelocity defines a radialvelocity measure from the CLI. It has to specify a reference code, radialvelocity quantity value (see introduction for the action on a scalar quantity with either a vector or scalar value),
and optionally it can specify an offset, which in itself has to be a radialvelocity. Allowable reference codes are: *LSRK LSRD BARY GEO TOPO GALACTO LGROUP CMB*.
Note that additional ones may become available. Check in CASA with:

```
print "\t----\t radialvelocity Ex 1 \t----"
print me.listcodes(me.radialvelocity())
#  Out[17]:
#{'extra': [],
# 'normal': ['LSRK', 'LSRD', 'BARY', 'GEO', 'TOPO', 'GALACTO',
# 'LGROUP', 'CMB']}
```

The radialvelocity quantity values should be given as velocity.

Arguments

Inputs		
rf	reference code	
	allowed:	string
	Default:	LSRK
v0	radial velocity	
	allowed:	any
	Default:	variant
off	optional offset radialvelocity measure	
	allowed:	record
	Default:	

Returns

record

Example

```
print "\t----\t radialvelocity Ex 2 \t----"
print me.radialvelocity('lsrk','20km/s')
#  Out[18]:
#{'m0': {'value': 20000.0, 'unit': 'm/s'},
# 'refer': 'LSRK',
# 'type': 'radialvelocity'}
```

---

measures.shift.html

measures.shift - Function

1.4.1 Shift a direction measure by an offset angle at a position angle.

Description

This method calculates the direction measure located at the specified offset angular amount along the specified position angle from the specified direction measure.

Arguments

Inputs		
v	The direction measure to shift, represented as a record.	
	allowed:	record
	Default:	
offset	The angular offset, represented as a quantity record or string.	
	allowed:	any
	Default:	variant 0deg
pa	Position angle of the offset, measured from the positive latitude axis through the positive longitude axis.	
	allowed:	any
	Default:	variant 0deg

Returns

record

Example

```
v = me.direction("J2000", "13:22:44", "-50.20.20")
# shift along 4 arcminues at a pa of 30 degrees.
offset = me.shift(v, offset="4arcmin", pa="30deg")
```

measures.uvw.html

measures.uvw - Function

1.4.1 define a uvw measure

Description

uvw defines a uvw measure from the CLI. It has to specify a reference code, uvw quantity values (see introduction for the action on a scalar quantity with either a vector or scalar value), and optionally it can specify an offset, which in itself has to be a uvw. Allowable reference codes are ITRF and the direction ones.
Note that additional ones may become available. Check in CASA with:

```
print "\t----\t uvw Ex 1 \t----"
print me.listcodes(me.uvw())
#{'normal': ['J2000', 'JMEAN', 'JTRUE', 'APP', 'B1950', 'BMEAN',
# 'BTRUE', 'GALACTIC', 'HADEC', 'AZEL', 'AZELSW', 'AZELNE',
# 'AZELGEO', 'AZELSWGEO', 'AZELNEGEO', 'JNAT', 'ECLIPTIC',
# 'MECLIPTIC', 'TECLIPTIC', 'SUPERGAL', 'ITRF', 'TOPO',
# 'ICRS'], 'extra': []}
```

The uvw quantity values should be either longitude (angle), latitude(angle) and height(length); or x,y,z (length). See quantity for possible angle formats.

Arguments

Inputs		
rf	reference code	
	allowed:	string
	Default:	ITRF
v0	longitude or x	
	allowed:	any
	Default:	variant
v1	latitude or y	
	allowed:	any
	Default:	variant
v2	height or z	
	allowed:	any
	Default:	variant
off	optional offset uvw measure	
	allowed:	record
	Default:	

Returns
record

Example

```
print "\t----\t uvw Ex 2 \t----"
print me.uvw('itrf','30deg','40deg','10m')
#{'m0': {'value': 0.52359877559829882, 'unit': 'rad'},
# 'm1': {'value': 0.6981317007977319, 'unit': 'rad'},
# 'm2': {'value': 9.9999999999999982, 'unit': 'm'},
# 'refer': 'ITRF',
# 'type': 'uvw'}
print me.doframe(me.epoch('utc','today'))
#True
print me.doframe(me.observatory('ALMA'))
#True
print me.doframe(me.direction('mars'))
#True
print me.measure(me.uvw('itrf','30deg','40deg','10m'), 'j2000')
#{'m0': {'value': 0.52321924738347259, 'unit': 'rad'},
```

```
# 'm1': {'value': 0.69813169995801672, 'unit': 'rad'},
# 'm2': {'value': 10.0, 'unit': 'm'},
# 'refer': 'J2000',
# 'type': 'uvw'}
```

---

measures.touvw.html

measures.touvw - Function

1.4.1 calculate a uvw measure from a baseline

Description

touvw calculates a uvw measure from a baseline. Note that the baseline does not have to be a proper *baseline*, but can be a series of positions (to call positions baselines see asbaseline) for speed reasons: operations are linear and can be done on positions, which are converted to baseline values at the end (with expand).
Whatever the reference code of the baseline, the returned *uvw* will be given in J2000. If the *dot* argument is given, that variable will be filled with a quantity array consisting of the time derivative of the uvw (note that only the sidereal rate is taken into account; not precession, earth tides and similar variations, which are much smaller). If the *xyz* variable is given, it will be filled with the quantity values of the uvw measure.
The values of the input baselines can be given as a quantity vector per x, y or z value.
uvw coordinates are calculated for a certain direction in the sky; hence the frame has to contain the direction for the calculation to work. Since the baseline and the sky rotate with respect of each other, the time should be specified as well.

Arguments

Outputs		
dot	uvw-dot (quantity array)	
	allowed:	record
	Default:	
xyz	uvw (quantity array)	
	allowed:	record
	Default:	
Inputs		
v	baseline measure	
	allowed:	record
	Default:	

Returns

record

Example

```
print "\t----\t touvw Ex 1 \t----"
print me.doframe(me.observatory('atca'))
#True
print me.doframe(me.source('1934-638'))
#True
print me.doframe(me.epoch('utc',qa.unit('today')))
#True
b=me.baseline('itrf','10m','20m','30m')
print me.touvw(b)
#{'dot': {'unit': 'm/s',
#         'value': [-0.0011912452908351659,
#                   -0.00098731747136827593,
#                   -0.00048769097314181744]},
# 'return': {'m0': {'value': -0.094777304811312649, 'unit': 'rad'},
#            'm1': {'value': -1.1509286139398101, 'unit': 'rad'},
#            'm2': {'value': 37.416573867739416, 'unit': 'm'},
#            'refer': 'J2000',
#            'type': 'uvw'},
# 'xyz': {'unit': 'm',
#         'value': [15.184026188402472,
#                   -1.4434256399579168,
#                   -34.166677788919138]}}
print me.getvalue(me.touvw(b))
#{'m0': {'value': -0.094777304811312649, 'unit': 'rad'},
# 'm1': {'value': -1.1509286139398101, 'unit': 'rad'},
# 'm2': {'value': 37.416573867739416, 'unit': 'm'}}
print me.getvalue(me.touvw(b))['m0']
#{'value': -0.094777304811312649, 'unit': 'rad'}

###Or when you are dealing with multiple antennas
####set the frame..i,e where, direction and when.
me.doframe(me.observatory('VLA'))
me.doframe(me.direction('J2000', '19h20m00', '20d10m00'))
me.doframe(me.epoch('utc', '2007/07/08/20:30:00'))
####antenna positions
ants=me.position('itrf',qa.quantity([3828763.11,3828746.55, 3828727.43],'m'), qa.quantity([4
###convert to baseline measures
```

```
bl=me.asbaseline(ants)
###convert to uvw
me.touvw(bl)

#{'dot': {'unit': 'm/s',
#         'value': array([ 181.25190155,  -73.29924893,  199.57974846,  181.25985238,
#        -73.29691498,  199.57339353,  181.2583565 ,  -73.29668498,
#        199.57276731])},
# 'return': {'m0': {'unit': 'rad',
#                   'value': array([ 2.21611194,  2.21610131,  2.21609887])},
#            'm1': {'unit': 'rad',
#                   'value': array([ 0.6984441 ,  0.69846521,  0.69846285])},
#            'm2': {'unit': 'm',
#                   'value': array([ 6364639.28758924,  6364639.27051283,  6364627.33064587]
#            'refer': 'J2000',
#            'type': 'uvw'},
# 'xyz': {'unit': 'm',
#         'value': array([-2931661.69632123,  3894141.52172208,  4092634.20894752,
#       -2931568.34776551,  3894103.64373003,  4092737.08879791,
#       -2931559.14911939,  3894111.22249941,  4092717.89890567])}}


####print the (n-1)n/2 baselines(u,v,w)
me.expand(me.touvw(bl)['return'])['xyz']
#{'unit': 'm',
# 'value': array([  93.34855573,  -37.87799205,  102.8798504 ,  102.54720184,
#        -30.29922267,   83.68995815,    9.19864612,    7.57876938,
#        -19.18989224])}
```

measures.expand.html

measures.expand - Function

1.4.1 expand n positions to n*(n-1)/2 baselines

Description

expand calculates the differences between a series of given measure values: it calculates baseline values from position values. The returned value is a measure, but the value of the optional output variable *xyz* will be set to an array of values.

Arguments

Outputs		
xyz	uvw (quantity array)	
	allowed:	record
	Default:	
Inputs		
v	measure (baseline, position or uvw measure)	
	allowed:	record
	Default:	

Returns

record

Example

```
print "\t----\t expand Ex 1 \t----"
b=me.baseline('itrf', qa.quantity([10, 20, 30], 'm'), qa.quantity([10, 20, 30], 'm'), qa.qua
print me.expand(b)
me.expand(b)

#{'return': {'m0': {'unit': 'rad',
#                    'value': array([ 0.78539816,  0.78539816,  0.78539816])},
#            'm1': {'unit': 'rad', 'value': array([ 0.,  0.,  0.])},
```

```
#            'm2': {'unit': 'm',
#                   'value': array([ 14.14213562,  28.28427125,  14.14213562])},
#            'refer': 'ITRF',
#            'type': 'baseline'},
# 'xyz': {'unit': 'm',
#         'value': array([ 10.,  10.,   0.,  20.,  20.,   0.,  10.,  10.,   0.])}}

print me.expand(b)['xyz']['value']

#[ 10.  10.   0.  20.  20.   0.  10.  10.   0.]
```

---

measures.earthmagnetic.html

measures.earthmagnetic - Function

1.4.1 define an earthmagnetic measure

Description

earthmagnetic defines an earthmagnetic measure from the CLI. It needs a reference code, earthmagnetic quantity values (see introduction for the action on a scalar quantity with either a vector or scalar value) if the reference code is not for a model, and optionally it can specify an offset, which in itself has to be a earthmagnetic. In general you specify a model (IGRF is the default and the only one known) and convert it to an explicit field. (See
http://fdd.gsfc.nasa.gov/IGRF.html
for information on the International Geomagnetic Reference Field). The earthmagnetic quantity values should be either longitude (angle), latitude(angle) and length(field strength); or x,y,z (field). See quantity for possible angle formats.

Arguments

Inputs		
rf	reference code	
	allowed:	string
	Default:	IGRF
v0	Field strength	
	allowed:	any
	Default:	variant
v1	longitude	
	allowed:	any
	Default:	variant
v2	latitude	
	allowed:	any
	Default:	variant
off	optional offset earthmagnetic measure	
	allowed:	record
	Default:	

Returns
record

Example

```
print "\t----\t earthmagnetic Ex 1 \t----"
print me.earthmagnetic('igrf')
#{'type': 'earthmagnetic', 'refer': 'IGRF', 'm1': {'value': 0.0, 'unit': 'nT'},
# 'm0': {'value': 6.1230317691118855e-23, 'unit': 'nT'},
# 'm2': {'value': 9.9999999999999995e-07, 'unit': 'nT'}}
print me.doframe(me.observatory('atca'))
print me.doframe(me.source('1934-638'))
print me.doframe(me.epoch('utc',qa.unit('today')))
print me.measure(me.earthmagnetic('igrf'), 'j2000')
#{'type': 'earthmagnetic', 'refer': 'J2000',
# 'm1': {'value': -8664.8767628222304, 'unit': 'nT'},
# 'm0': {'value': 50544.054410564473, 'unit': 'nT'},
# 'm2': {'value': 1799.5131920958615, 'unit': 'nT'}}
```

---

measures.baseline.html

measures.baseline - Function

1.4.1 define a baseline measure

Description

baseline defines a baseline measure from the CLI. It has to specify a reference code, baseline quantity values (see introduction for the action on a scalar quantity with either a vector or scalar value, and when a vector of quantities is given), and optionally it can specify an offset, which in itself has to be a baseline. Allowable reference codes are ITRF and the direction ones.
Note that additional ones may become available. Check in CASA with:

```
print "\t----\t baseline Ex 1 \t----"
print me.listcodes(me.baseline())
#{'normal': ['J2000', 'JMEAN', 'JTRUE', 'APP', 'B1950', 'BMEAN', 'BTRUE',
# 'GALACTIC', 'HADEC', 'AZEL', 'AZELSW', 'AZELNE', 'AZELGEO', 'AZELSWGEO',
# 'AZELNEGEO', 'JNAT', 'ECLIPTIC', 'MECLIPTIC', 'TECLIPTIC', 'SUPERGAL',
# 'ITRF', 'TOPO', 'ICRS'], 'extra': []}
```

The baseline quantity values should be either longitude (angle), latitude(angle) and height(length); or x,y,z (length). See quantity for possible angle formats.

Arguments

Inputs		
rf	reference code	
	allowed:	string
	Default:	ITRF
v0	longitude or x	
	allowed:	any
	Default:	variant
v1	latitude or y	
	allowed:	any
	Default:	variant
v2	height or z	
	allowed:	any
	Default:	variant
off	optional offset baseline measure	
	allowed:	record
	Default:	

Returns
record

Example

```
print "\t----\t Ex 2 \t----"
print me.baseline('itrf','30deg','40deg','10m')
#{'m0': {'value': 0.52359877559829882, 'unit': 'rad'},
# 'm1': {'value': 0.6981317007977319, 'unit': 'rad'},
# 'm2': {'value': 9.9999999999999982, 'unit': 'm'},
# 'refer': 'ITRF',
# 'type': 'baseline'}
print me.doframe(me.observatory('atca'))
print me.doframe(me.source('1934-638'))
print me.doframe(me.epoch('utc',qa.unit('today')))
print me.measure(me.baseline('itrf','30deg','40deg','10m'), 'J2000')
#{'m0': {'value': 0.58375325605991979, 'unit': 'rad'},
# 'm1': {'value': 0.69758519780286155, 'unit': 'rad'},
# 'm2': {'value': 9.9999999999999964, 'unit': 'm'},
# 'refer': 'J2000',
```

```
# 'type': 'baseline'}
```

---

measures.asbaseline.html

measures.asbaseline - Function

1.4.1 define a baseline from a position measure

Description

asbaseline converts a position measure into a baseline measure. No actual baseline is calculated, since operations can be done on positions, with subtractions to obtain baselines at a later stage.

Arguments

Inputs		
pos	position measure	
	allowed:	record
	Default:	

Returns

record

Example

```
print "\t----\t asbaseline Ex 1 \t----"

####An example of getting baselines with 3 antenna positions
#### Define the frame ; where, which-direction and when
me.doframe(me.observatory('VLA'))
me.doframe(me.direction('J2000', '19h20m00', '20d10m00'))
me.doframe(me.epoch('utc', '2007/07/08/20:30:00'))

##antenna position
ants=me.position('itrf',qa.quantity([3828763.11,3828746.55, 3828727.43],'m'), qa.quantity([4
print ants
#{'type': 'position', 'refer': 'ITRF', 'm1': {'value': array([ 0.92031276,  0.92031276,  0.9
#'m0': {'value': array([ 0.11504897,  0.11508633,  0.1150838 ]), 'unit': 'rad'},
```

```
#'m2': {'value': array([ 6364639.28758924,  6364639.27051283,  6364627.33064587]), 'unit': '

bl=me.asbaseline(ants)
print bl
#{'type': 'baseline', 'refer': 'J2000', 'm1': {'value': array([ 0.92068328,  0.92068326,  0.
#'m0': {'value': array([-2.08658811, -2.08655073, -2.08655326]), 'unit': 'rad'},
#'m2': {'value': array([ 6364639.28758924,  6364639.27051283,  6364627.33064587]), 'unit': '

me.expand(bl)

#{'return': {'m0': {'unit': 'rad',
#                   'value': array([-0.51637894, -0.36575235,  1.50036599])},
#            'm1': {'unit': 'rad',
#                   'value': array([-0.00060966,  0.00302388,  0.02206414])},
#            'm2': {'unit': 'm',
#                   'value': array([ 143.98943974,  135.78652583,   22.58992696])},
#            'refer': 'J2000',
#            'type': 'baseline'},
# 'xyz': {'unit': 'm',
#         'value': array([  1.25215025e+02,  -7.10925354e+01,  -8.77850493e-02,
#         1.26804339e+02,  -4.85640980e+01,   4.10601842e-01,
#         1.58931410e+00,   2.25284374e+01,   4.98386892e-01])}}
```

---

measures.listcodes.html

measures.listcodes - Function

1.4.1 get known reference code names (list indices do not necessarily correspond to enumeration indices)

Description

listcodes will produce the known reference codes for a specified measure type. It will return a record with two entries. The first is a string vector of all normal codes; the second a string vector (maybe empty) with all extra codes (like planets). NOTE: Synonyms and different code groups may be present in the code name lists. The indices in these lists therefore do not necessarily correspond to the internal CASA enumeration indices.

Arguments

Inputs		
ms	the measure type for which to list	
	allowed:	record
	Default:	

Returns
record

Example

```
print "\t----\t listcodes Ex 1 \t----"
# Generate some direction
# Note that an empty or non-specified reference code will produce the
# measure with the default code for that measure type
a=me.direction()
print me.getref(a)
#'J2000'
print me.ismeasure(a)
#True
```

```
# Get the known reference codes for direction
print me.listcodes(a)
#{'normal': ['J2000', 'JMEAN', 'JTRUE', 'APP', 'B1950', 'BMEAN',
#'BTRUE', 'GALACTIC', 'HADEC', 'AZEL', 'AZELSW', 'AZELNE', 'AZELGEO',
#'AZELSWGEO', 'AZELNEGEO', 'JNAT', 'ECLIPTIC', 'MECLIPTIC',
#'TECLIPTIC', 'SUPERGAL', 'ITRF', 'TOPO', 'ICRS'],
#'extra': ['MERCURY', 'VENUS', 'MARS', 'JUPITER', 'SATURN', 'URANUS',
#'NEPTUNE', 'PLUTO', 'SUN', 'MOON', 'COMET']}
```

---

measures.measure.html

measures.measure - Function

1.4.1 convert a measure to another reference

Description

measure converts measures (epoch, direction etc.) from one reference to another. It will, for instance, convert a direction from J2000 to AZEL representation.
Its arguments are a measure, an output reference code (see the individual measures for the allowable codes (direction, position, epoch, frequency, doppler, radialvelocity, baseline, uvw, earthmagnetic)), and an optional offset of the same type as the main measure. The offset will be subtracted from the result before it is returned.
In some cases (see the individual measures for when), more information than just a reference code is necessary. E.g. the above example of a conversion to AZEL, needs to know for when, and where on Earth we want it. This information is stored in a reference frame. Measures are set in the reference frame with the doframe function. The frame is tool wide.
IMPORTANT NOTE:
To get an accurate conversion of solar system objects direction to a celestial frame, one should convert to AZEL or HADEC before to get parallax accounted for. Thus if you want to get the moon's position in J2000..one would do it in 2 stages
i.e (after setting the appropriate frames)
moonazel=me.measure(me.direction('moon'), 'AZELGEO')
moonJ2000=me.measure(moonazel, 'J2000')

Arguments

Inputs		
v	measure to be converted	
	allowed:	record
	Default:	
rf	output reference code	
	allowed:	string
	Default:	
off	optional output offset measure	
	allowed:	record
	Default:	

Returns
record

Example

```
print "\t----\t measure Ex 1 \t----"
a = me.epoch('utc','today')          # a time
print a
#{'m0': {'value': 54054.872957673608, 'unit': 'd'},
# 'refer': 'UTC',
# 'type': 'epoch'}
print me.doframe(me.source('1934-638'))
print me.measure(a, 'tai') # convert to IAT
#{'m0': {'value': 54054.873339618054, 'unit': 'd'},
# 'refer': 'TAI',
# 'type': 'epoch'}
print me.doframe(a) # set time in frame
#True
print me.doframe(me.observatory('ALMA')) # set position in frame
#True
b=me.direction('j2000', qa.toangle('0h'), '-30deg')  # a direction
print b
#{'m0': {'value': 0.0, 'unit': 'rad'},
# 'm1': {'value': -0.52359877559829882, 'unit': 'rad'},
# 'refer': 'J2000',
# 'type': 'direction'}
print me.measure(b, 'azel') # convert to AZEL
#{'m0': {'value': 1.9244096810822324, 'unit': 'rad'},
# 'm1': {'value': 0.76465385681363052, 'unit': 'rad'},
# 'refer': 'AZEL',
# 'type': 'direction'}
print qa.angle(me.getvalue(me.measure(b,'azel'))['m0'])     # show as angles
#['+110.15.38']
print qa.angle(me.getvalue(me.measure(b,'azel'))['m1'])
#['+043.48.41']


Another example:

print "\t----\t measure Ex 2 \t----"
# Fill the frame with necessary information
```

```
print me.doframe(me.epoch('utc','today'))
#True
print me.doframe(me.observatory('ALMA'))
#True
print me.doframe(me.direction('mars'))
#True
a=qa.unit('1GHz')
print a
#{'value': 1.0, 'unit': 'GHz'}
m=me.frequency('lsrk',qa.quantity(qa.getvalue(a),qa.getunit(a)))
print m
#{'m0': {'value': 1000000000.0, 'unit': 'Hz'},
# 'refer': 'LSRK',
# 'type': 'frequency'}
print me.measure(m,'lsrd')
#{'m0': {'value': 1000001766.3928765, 'unit': 'Hz'},
# 'refer': 'LSRD',
# 'type': 'frequency'}
```

---

measures.doframe.html

measures.doframe - Function

1.4.1 save a measure as frame reference

Description

doframe will set the measure specified as part of a frame.
If conversion from one type to another is necessary, with the measure function, the following frames should be set if one of the reference types involved in the conversion is as in the following lists.
Epoch
UTC TAI LAST position LMST position GMST1 GAST UT1 UT2 TDT TCG TDB TCD
Direction
J2000 JMEAN epoch JTRUE epoch APP epoch B1950 BMEAN epoch BTRUE epoch GALACTIC HADEC epoch position AZEL epoch position SUPERGALACTIC ECLIPTIC MECLIPTIC epoch TECLIPTIC epoch PLANET epoch [position]
Position
WGS84 ITRF
Radial Velocity
LSRK direction LSRD direction BARY direction GEO direction epoch TOPO direction epoch position GALACTO direction
Doppler
RADIO OPTICAL Z RATIO RELATIVISTIC BETA GAMMA
Frequency
REST direction radialvelocity LSRK direction LSRD direction BARY direction GEO direction epoch TOPO direction epoch position GALACTO

Arguments

Inputs		
v	measure to be set in frame	
	allowed:	record
	Default:	

Returns
bool

Example

```
print "\t----\t doframe Ex 1 \t----"
a = me.epoch('utc', 'today')  # a time
print a
#{'m0': {'value': 54054.91671484954, 'unit': 'd'},
# 'refer': 'UTC',
# 'type': 'epoch'}
print me.doframe(a) # set time in frame
#True
```

---

measures.framenow.html

measures.framenow - Function

1.4.1 set the active frame time at now

Description

framenow will fill the active frame time with the current date and time. The different frame values necessary are described in the doframe function

Arguments

Inputs

Returns
bool

Example

```
print "\t----\t framenow Ex 1 \t----"
print me.framenow() # specify now as frame reference
#True
print me.showframe()         # and show the current frame
#'Frame: Epoch: 54054::22:01:42.2880'
```

measures.showframe.html

measures.showframe - Function

1.4.1 show the currently active frame reference

Description

showframe will display the currently active reference frame values on the terminal. The different frame values necessary are described in the doframe function. The frame is displayed on the terminal using the formatting as done for the show function.

Arguments

Inputs

Returns

string

Example

```
print "\t----\t showframe Ex 1 \t----"
print me.doframe(me.epoch('utc','today')) # specify now as frame reference
#T
print me.showframe() # and show the current frame
#'Frame: Epoch: 54054::22:01:42.2880'
```

---

measures.toradialvelocity.html

measures.toradialvelocity - Function

1.4.1 convert a doppler type value to a real radial velocity

Description

toradialvelocity will convert a Doppler type value (e.g. in radio mode) to a real radialvelocity. The type of velocity (e.g. LSRK) should be specified

Arguments

Inputs		
rf	radial velocity reference type	
	allowed:	string
	Default:	
v0	doppler value measure	
	allowed:	record
	Default:	

Returns
record

Example

```
print "\t----\t toradialvelocity Ex 1 \t----"
a = me.doppler('radio','0.4')
print a
#  Out[4]:
#{'m0': {'value': 119916983.2, 'unit': 'm/s'},
# 'refer': 'RADIO',
# 'type': 'doppler'}
print me.toradialvelocity('topo',a)
#{'m0': {'value': 141078803.7647059, 'unit': 'm/s'},
# 'refer': 'TOPO',
# 'type': 'radialvelocity'}
```

measures.tofrequency.html

measures.tofrequency - Function

1.4.1 convert a doppler type value to a frequency

Description

tofrequency will convert a Doppler type value (e.g. in radio mode) to a frequency. The type of frequency (e.g. LSRK) and a rest frequency (either as a frequency quantity (e.g. qa.constants('HI')) or a frequency measure (e.g. me.frequency('rest','5100MHz')) should be specified

Arguments

Inputs		
rf	frequency reference type	
	allowed:	string
	Default:	
v0	doppler measure value	
	allowed:	record
	Default:	
rfq	rest frequency (frequency measure or freuency quantity)	
	allowed:	record
	Default:	

Returns

record

Example

```
print "\t----\t tofrequency Ex 1 \t----"
a=me.doppler('radio','0.4')
print a
#{'m0': {'value': 119916983.2, 'unit': 'm/s'},
# 'refer': 'RADIO',
# 'type': 'doppler'}
```

```
print me.tofrequency('lsrk',a,qa.constants('HI'))
#{'m0': {'value': 852243451.07159996, 'unit': 'Hz'},
# 'refer': 'LSRK',
# 'type': 'frequency'}
```

---

measures.todoppler.html

measures.todoppler - Function

1.4.1 convert a frequency or radialvelocity measure to a doppler measure

Description

todoppler will convert a radialvelocity measure or a frequency measure to a doppler measure. In the case of a frequency, a rest frequency has to be specified. The type of doppler wanted (e.g. RADIO) has to be specified.

Arguments

Inputs		
rf	doppler reference type	
	allowed:	string
	Default:	
v0	radial velocity or frequency measure	
	allowed:	record
	Default:	
rfq	rest frequency (frequency measure or frequency quantity)	
	allowed:	any
	Default:	variant

Returns

record

Example

```
print "\t----\t todoppler Ex 1 \t----"
f = me.frequency('lsrk','1410MHz')     # specify a frequency
print f
#{'m0': {'value': 1410000000.0, 'unit': 'Hz'},
# 'refer': 'LSRK',
```

```
# 'type': 'frequency'}
print me.todoppler('radio', f, qa.constants('HI')) # give doppler, using HI rest
#{'m0': {'value': 2196249.8401180855, 'unit': 'm/s'},
# 'refer': 'RADIO',
# 'type': 'doppler'}
```

---

measures.torestfrequency.html

measures.torestfrequency - Function

1.4.1 convert a frequency and doppler measure to a rest frequency

Description

torestfrequency will convert a frequency measure and a doppler measure (e.g. obtained from another spectral line with a known rest frequency) to a rest frequency.

Arguments

Inputs		
v0	frequency reference type	
	allowed:	record
	Default:	
d0	doppler measure value	
	allowed:	record
	Default:	

Returns

record

Example

```
print "\t----\t torestfrequency Ex 1 \t----"
dp = me.doppler('radio', '2196.24984km/s')  # a measured doppler speed
print dp
#{'m0': {'value': 2196249.8399999999, 'unit': 'm/s'},
# 'refer': 'RADIO',
# 'type': 'doppler'}
f = me.frequency('lsrk','1410MHz')    # a measured frequency
print f
#{'m0': {'value': 1410000000.0, 'unit': 'Hz'},
# 'refer': 'LSRK',
```

```
# 'type': 'frequency'}
print me.torestfrequency(f, dp)                    # the corresponding rest frequency
#{'m0': {'value': 1420405751.7854364, 'unit': 'Hz'},
# 'refer': 'REST',
# 'type': 'frequency'}
```

---

measures.rise.html

measures.rise - Function

1.4.1 get rise and set sidereal time

Description

rise will give the rise/set hour-angles of a source. It needs the position in the frame, and a time. If the latter is not set, the current time will be used.

Arguments

Inputs		
crd	direction of source (direction measure)	
	allowed:	any
	Default:	variant
ev	elevation angle limit	
	allowed:	any
	Default:	variant 0.0deg

Returns

record

Example

```
# NOT IMPLEMENTED
print "\t----\t rise Ex 1 \t----"
print me.rise(me.direction('sun'))
#[rise=[value=267.12445, unit=deg], set=[value=439.029964, unit=deg]]
print qa.form.long(me.rise(me.direction('sun')).rise)
#17:48:29.868
#
```

measures.riseset.html

measures.riseset - Function

1.4.1 get rise and set times

Description

rise will give the rise/set times of a source. It needs the position in the frame, and a time. If the latter is not set, the current time will be used. The returned value is a record with a 'solved' field, which is F if the source is always below or above the horizon. In that case the rise and set fields will all have a string value. The record also returns a rise and set record, with 'last' and 'utc' fields showing the rise and set times as epochs.

Arguments

Inputs		
crd	direction of source (direction measure)	
	allowed:	any
	Default:	variant
ev	elevation limit	
	allowed:	any
	Default:	variant 0.0deg

Returns

record

Example

```
# NOT IMPLEMENTED
print "\t----\t riseset Ex 1 \t----"
print me.riseset(me.direction('sun'))
#[solved=T,
# rise=[last=[type=epoch, refer=LAST, m0=[value=0.0731388605, unit=d]],
#       utc=[type=epoch, refer=UTC, m0=[value=52085.8964, unit=d]]],
# set=[last=[type=epoch, refer=LAST, m0=[value=0.455732593, unit=d]],
```

```
#       utc=[type=epoch, refer=UTC, m0=[value=52086.2779, unit=d]]]]
print me.riseset(me.direction('sun'), qa.unit('80deg'))
#[solved=F,
# rise=[last=below, utc=below],
# set=[last=below, utc=below]]
print qa.form.long(me.riseset(me.direction('sun')).rise.utc.m0)
#21:30:47.439
#
```

---

measures.posangle.html

measures.posangle - Function

1.4.1 get position angle of two directions

Description

posangle will give the position angle from a direction to another. I.e. the angle in a direction between the direction to the North pole and the other direction. The posiation angle is calculated in the frame of the first argument. m2 is thus converted to the frame of m1 before calculating the position angle.

Arguments

Inputs		
m1	direction of source (direction measure)	
	allowed:	record
	Default:	
m2	direction of other source (direction measure)	
	allowed:	record
	Default:	

Returns

record

Example

```
print "\t----\t posangle Ex 1 \t----"
a=me.direction('j2000','0deg','70deg')
b=me.direction('j2000','0deg','80deg')
print me.posangle(a,b)
#{'value': -0.0, 'unit': 'deg'}
print me.separation(a,b)
#{'value': 9.9999999999999893, 'unit': 'deg'}
tim=me.epoch('utc','today')
print me.doframe(tim)
```

```
#True
pos=me.observatory('ATCA')
print me.doframe(pos)
#True
print me.posangle(a,b)
#{'value': -0.0, 'unit': 'deg'}

###Example of how to calculate the parallactic angle of a given direction on thesky.

###set the frames and epoch
```

---

measures.separation.html

measures.separation - Function

1.4.1 get separation angle between two directions

Description

separation will give the separation of a direction from another as an angle.

Arguments

Inputs		
m1	direction of source (direction measure)	
	allowed:	record
	Default:	
m2	direction of other source (direction measure)	
	allowed:	record
	Default:	

Returns

record

Example

```
print "\t----\t separation Ex 1 \t----"
a=me.direction('j2000','0deg','70deg')
b=me.direction('j2000','0deg','80deg')
print me.separation(a,b)
#{'value': 9.9999999999999893, 'unit': 'deg'}
tim = me.epoch('utc','today')              # set the time
print me.doframe(tim)
#True
pos = me.observatory('ATCA')               # set where
print me.doframe(pos)
#True
c=me.measure(b,'azel')                     # try with different type
```

```
print me.separation(a,c)
#{'value': 10.000000000062277, 'unit': 'deg'}

### the example below is how to calculate
### the parallactic angle
me.doframe(me.epoch('utc','2015/06/30/19:30:40')))
me.doframe(me.observatory('ALMA'))
mydir = me.direction('J2000','17h28m00','-28d00m00' )
#convert direction to AZEL
mydirazel=me.measure(mydir, 'AZEL')
hadecpol=me.direction('HADEC', '00h00m00', '90d00m00')
### no need to convert north pole direction to AZEL
### as it will coverted to the frame of mydirazel
parAngle=me.posangle(mydirazel, hadecpol)
```

---

measures.addxvalue.html

measures.addxvalue - Function

1.4.1 get some additional measure information

Description

addxvalue will give some additional information about some measures as a vector of quantities. It is used internally to get the rectangular coordinates of measures that are normally given in angles. The casual user will probably in general not interested in this function.

Arguments

Inputs		
a	measures for which extra information is to be gotten	
	allowed:	record
	Default:	

Returns

record

Example

```
print "\t----\t addxvalue Ex 1 \t----"
a=me.observatory('atca')
print a
#{'m0': {'value': 2.6101423190348916, 'unit': 'rad'},
# 'm1': {'value': -0.5261379196128062, 'unit': 'rad'},
# 'm2': {'value': 6372960.2577234386, 'unit': 'm'},
# 'refer': 'ITRF',
# 'type': 'position'}
print me.addxvalue(a)
#{'value': [-4750915.8370000012, 2792906.1819999996, -3200483.747], 'unit': 'm'}
print me.addxvalue(me.epoch('utc','today'))
#{}
```

measures.type.html

measures.type - Function

1.4.1 type of tool

Description

type will return the tool name.

Arguments

Inputs

Returns

string

Example

```
print "\t----\t type Ex 1 \t----"
print me.type()
#'measures'
```

---

measures.done.html

measures.done - Function

1.4.1 free resources used by tool.

Description

In general you will not want to call this method. It removes and then recreates the default measures tool.

Arguments

Inputs

Returns
bool

Example

```
print "\t----\t done Ex 1 \t----"
print me.done()
#True
```

---

measures.ismeasure.html

measures.ismeasure - Function

1.4.1 Check if measure

Description

Checks if the operand is a correct measure

Arguments

Inputs		
v	value to be tested	
	allowed:	record
	Default:	

Returns

bool

Example

```
print "\t----\t ismeasure Ex 1 \t----"
x=me.epoch('utc','today')
print x
#{'m0': {'value': 54056.043754386577, 'unit': 'd'},
# 'refer': 'UTC',
# 'type': 'epoch'}
print me.ismeasure(x)
#True
y=me.getvalue(x)
print y
#{'m0': {'value': 54056.043754386577, 'unit': 'd'}}
print me.ismeasure(y)
#False
print "Last example, exiting!"
exit()
```

quanta-Module.html

1.5 quanta - Module

Units and quantities handling

Description *Introduction*

A quantity is a value with a unit. For example, '5km/s', or '20Jy/pc2'. This module (the `quanta` module) enables you to create and manipulate such quantities. The types of functionality provided are:

- *Conversion* - Conversion of quantities to different units
- *Calculation* - Calculations with quantities

The Quanta `tool` manipulates quantities. A quantity is stored as a record with two fields. These fields are named 'value' and 'unit'. As well as simple scalar quantites, one can also create quantities as vectors or arrays. For example, you may have a vector of values, which all have the same unit - there is no need to store a copy of the unit for each value. Access to the individual fields of a quantum should always be by using the *getvalue* and *getunit* methods, especially since the internal names can change or be not accessable at some stage.

Example

```
"""
#
print "\t----\t Module Ex 1 \t----"
print qa.quantity(5.4, 'km/s')
#{'value': 5.4000000000000004, 'unit': 'km/s'}
q1 = qa.quantity([8.57132661e+09, 1.71426532e+10], 'km/s')
print qa.convert(q1, 'pc/h');
#{'value': array([ 1.,  2.]), 'unit': 'pc/h'}
#
"""
```

In the first example, we make a simple scalar quantity. You can see that the quantity (which is actually a record) has fields 'value' and 'unit'.

In the second example, we make a vector quantity and then convert it from units of km/s to pc/h.

Example

```
"""
#
print "\t----\t Module Ex 2 \t----"
print qa.quantity('5.4km/s')
#{'value': 5.4000000000000004, 'unit': 'km/s'}
print qa.quantity(qa.unit('5.4km/s'))
#{'value': 5.4000000000000004, 'unit': 'km/s'}
#
"""
```

In the first example, the value and unit were combined into one string (just saves a bit of typing). The second example shows that the function `unit` is an alias for `quantity`, and that you can create a quantity from another quantity.

Example

```
"""
#
print "\t----\t Module Ex 3 \t----"
q1 = qa.unit("5s 5.4km/s")
print len(q1)
#2
print q1['*0']
# {'unit': 's', 'value': 5.0}
print q1['*1']
# {'unit': 'km/s', 'value': 5.4000000000000004}
#
```

Here we make a vector quantity by using the string vector. You can see that the resultant quantity record is of length 2 and that each field of that vector quantity is a scalar quantity. So you see that 'q1' itself does not have fields 'value' and 'unit', only the elements of 'q1' have that.

Example

```
"""
#
print "\t----\t Module Ex 4 \t----"
q1 = qa.unit('5km');
q2 = qa.unit('200m');
print qa.canonical(qa.add(q1,q2))
#{'value': 5200.0, 'unit': 'm'}
#
"""
```

Here we make two quantities with consistent but different units, add them together and then convert the result to canonical units.

Example

```
"""
#
print "\t----\t Module Ex 5 \t----"
q1 = qa.quantity('6rad');
q2 = qa.quantity('3deg');
print qa.compare(q1,q2)
#True
print qa.compare(q1,qa.unit('3km'))
#False
#
"""
```

Here we compare the dimensionality of the units of two quantities.

Constants, time and angle formatting

If you would like to see all the possible constants known to the Quanta `tool` you can issue the command `print qa.map('const')`. You can get the value of any constant in that list with a command such as

```
"""
#
print "\t----\t Module Ex 6 \t----"
boltzmann = qa.constants('k')
print 'Boltzmann constant is ', boltzmann
#Boltzmann constant is  {'value': 1.3806577987510647e-23, 'unit': 'J/K'}
#
"""
```

There are some extra handy ways you can manipulate strings when you are dealing with times or angles. The following list shows special strings and string formats which you can input to the `quantity` function. Something in square brackets is optional. There are examples after the list.

- time: [+-]hh:mm:ss.t... – This is the preferred time format (trailing fields can be omitted)
- time: [+-]hhHmmMss.t..[S] – This is an alternative time format (HMS case insensitive, trailing second fields can be omitted)
- angle: [+-]dd.mm.ss.t.. – This is the preferred angle format (trailing fields after second priod can be omitted; dd.. is valid)
- angle: [+-]ddDmmMss.t...[S] – This is an alternative angle format (DMS case insensitive, trailing fields can be omitted after M)
- today – The special string "today" gives the UTC time at the instant the command was issued.

- today/time – The special string “today” plus the specified time string gives the UTC time at the specified instant
- yyyy/mm/dd[/time] – gives the UTC time at the specified instant
- dd[-]mmm[-][cc]yy[/time] – gives the UTC time at the specified instant in calendat style notation (23-jun-1999)

Note that the standard unit for degrees is 'deg', and for days 'd'. Formatting is done in such a way that it interprets a 'd' as degrees if preceded by a value without a period and if any value following it is terminated with an 'm'. In other cases 'days' are assumed. Here are some examples.

```
"""
#
print "\t----\t Module Ex 7 \t----"
print qa.quantity('today')
#{'value': 54178.87156457176, 'unit': 'd'}
print qa.quantity('5jul1998')
#{'value': 50999.0, 'unit': 'd'}
print qa.quantity('5jul1998/12:')
#{'value': 50999.5, 'unit': 'd'}
print qa.quantity('-30.12.2')
#{'value': -30.200555555555557, 'unit': 'deg'}
print qa.quantity('2:2:10')
#{'value': 30.541666666666668, 'unit': 'deg'}
print qa.unit('23h3m2.2s')
#{'value': 345.75916666666666, 'unit': 'deg'}
#
"""
```

Angles and times can often be used interchangeably. Special functions (`qa.totime()` and `qa.toangle()`) are available to make them in the right units for the purpose. E.g. `qa.sin(time)` gives an error, whereas `qa.sin(qa.toangle(time))` works ok. See the map function for pre-defined units.

```
"""
#
print "\t----\t Module Ex 8 \t----"
a = qa.quantity('today');                    # 1
print a
#{'value': 54178.871564641202, 'unit': 'd'}
b = qa.toangle(a);                           # 2
print b
#{'value': 340415.88977452344, 'unit': 'rad'}
print qa.angle(qa.norm(qa.toangle(a)));      # 3
#-046.14.12
```

```
print qa.angle(qa.norm(qa.toangle(a), 0));    # 4
#+313.45.48
print qa.sub('today',a);                      # 5
#{'value': 1.1576048564165831e-08, 'unit': 'd'}
#
print "Last example! Exiting ..."
exit()
"""
```

1. Get the time now
2. Get the time as an angle
3. Get the time as a normalised angle (-pi to +pi) and show as dms
4. Get the time as a normalised angle (0 to 2pi) and show as dms
5. Get time since creation of a

quanta-Tool.html

1.5.1 quanta - Tool

quanta tool handles units and quantities
Requires:

Synopsis

Methods

convertfreq	convert a frequency quantity to another unit
convertdop	convert a doppler velocity quantity to another unit
quantity	make a quantity from a string or from a numeric value and a unit string
getvalue	get the internal value of a quantity
getunit	get the internal unit of a quantity
canonical	get canonical value of quantity
canon	get canonical value of quantity
convert	convert a quantity to another unit
define	define a new unit name
map	list known unit names and constants
maprec	create record containing list of known unit names and constants
fits	define some FITS units
angle	show an angle as a formatted string
time	show a time (or date) as a formatted string
add	add quantities
sub	subtract quantities
mul	multiply quantities
div	divides quantities
neg	negate quantities
norm	normalise angle
le	compare quantities
lt	compare quantities
eq	compare quantities
ne	compare quantities
gt	compare quantities
ge	compare quantities
sin	sine of quantity
cos	cosine of quantity
tan	tangent of quantity
asin	arcsine of quantity
acos	arccosine of quantity
atan	arctangent of quantity
atan2	arctangent of two quantity
abs	absolute value of quantity
ceil	ceil value of quantity

floor	floor value of quantity
log	logarithm of quantity
log10	logarithm of quantity
exp	exponential of quantity
sqrt	square root of quantity
compare	compare dimensionality of units
check	check for proper unit string
checkfreq	check for proper frequency unit
pow	raise quantity to power
constants	get a constant
isangle	check if valid angle or time quantity
totime	convert an angle (or a time) to a time
toangle	convert a time (or an angle) to an angle
splitdate	split a date/time into a record
tos	convert quantity to string
type	type of tool
done	Free resources used by tool. Current implementation ignores input parameter, does nothing ar
unit	quantity from value v and unit string
isquantity	Check if quantity
setformat	set format for output of numbers. (NOT IMPLEMENTED YET!)
getformat	get current output format (NOT IMPLEMENTED YET!)
formxxx	Format a quantity using given format, allowed are hms, dms, deg, rad, +deg.

quanta.convertfreq.html

quanta.convertfreq - Function

1.5.1 convert a frequency quantity to another unit

Description

convertfreq converts a frequency quantity to another unit.

Arguments

Inputs		
v	quantity to convert	
	allowed:	variant
	Default:	1.0
outunit	unit to convert to	
	allowed:	string
	Default:	Hz

Returns

record

Example

```
"""
#
print "\t----\t convertfreq Ex 1 \t----"
print qa.convertfreq('5GHz','cm')
#{'value': 5.9958491599999997, 'unit': 'cm'}
print qa.convertfreq('5cm','GHz')
#{'value': 5.9958491599999997, 'unit': 'GHz'}
#
"""
```

quanta.convertdop.html

quanta.convertdop - Function

1.5.1 convert a doppler velocity quantity to another unit

Description

convertfreq converts a velocity quantity to another unit. Units are either velocity or dimensionless.

Arguments

Inputs		
v	quantity to convert	
	allowed:	variant
	Default:	0.0
outunit	unit to convert to	
	allowed:	string
	Default:	km/s

Returns

record

Example

```
"""
#
print "\t----\t convertdop Ex 1 \t----"
print qa.convertdop('1','km/s')
#{'value': 299792.45799999998, 'unit': 'km/s'}
print qa.convertdop('10km/s','1')
#{'value': 3.3356409519815205e-05, 'unit': '1'}
#
"""
```

quanta.quantity.html

quanta.quantity - Function

1.5.1 make a quantity from a string or from a numeric value and a unit string

Description

quantity makes a quantity from a string, or from a value and a string. Note that a function unit exists which is a synonym for quantity. If only a string is given, it can be a scalar string. The result will be a scalar quantity.
If a numeric value and a unit string are given, the numeric value can be any numeric type, and can also be a vector of numeric values. print qa.map() to get a list of recognized units. 'd' is usually days, but can be degrees (see example).

Arguments

Inputs		
v	quantity or numeric or string to convert to quantity	
	allowed:	variant
	Default:	
unitname	unit string if v numeric	
	allowed:	string
	Default:	

Returns
record

Example

```
"""
#
print "\t----\t quantity Ex 1 \t----"
tu = qa.quantity('1Jy') # make quantity
print tu
#{'value': 1.0, 'unit': 'Jy'}
print qa.quantity(tu) # also accepts a quantity
```

```
#{'value': 1.0, 'unit': 'Jy'}
tu = qa.unit('1Jy') # make quantity with synonym
print tu
#{'value': 1.0, 'unit': 'Jy'}
print qa.quantity(-1.3, 'Jy') # make quantity with separate value
#{'value': -1.3, 'unit': 'Jy'}
q1 = qa.quantity([8.57132661e+09, 1.71426532e+10], 'km/s') # Composite unit
print q1
#{'value': array([  8.57132661e+09,   1.71426532e+10]), 'unit': 'km/s'}
q = qa.quantity('5d'); print q
#{'value': 5.0, 'unit': 'd'}            # d = days
q = qa.quantity('5 d'); print q
#{'value': 5.0, 'unit': 'd'}            # even if there's a space, as of 5/28/09
q = qa.quantity('5d30m'); print q
#{'value': 5.5, 'unit': 'deg'}          # Unless followed by an m!
qa.quantity('5d30s')                    # WRONG
# {'unit': 'd30s', 'value': 5.0}        # I told you...
qa.quantity('5d0m30s')                  # OK
# {'unit': 'deg', 'value': 5.0083333333333337}
"""
```

---

quanta.getvalue.html

quanta.getvalue - Function

1.5.1 get the internal value of a quantity

Description

getvalue returns the internal value of a quantity. It also can handle an array of quantities.

Arguments

Inputs		
v	quantity	
	allowed:	variant
	Default:	

Returns

doubleArray

Example

```
"""
#
print "\t----\t getvalue Ex 1 \t----"
tu = qa.quantity(-1.3, 'Jy')         # make quantity
print tu
#{'value': -1.3, 'unit': 'Jy'}
print qa.getvalue(tu)
#-1.3
print qa.getunit(tu)
#Jy
a = qa.quantity([3,5],'cm')
print a
#{'value': array([ 3.,  5.]), 'unit': 'cm'}
print qa.getvalue(a)
```

```
#[3.0, 5.0]
#
"""
```

---

quanta.getunit.html

quanta.getunit - Function

1.5.1 get the internal unit of a quantity

Description

getunit returns the internal unit string of a quantity

Arguments

Inputs		
v	quantity	
	allowed:	variant
	Default:	

Returns

string

Example

```
"""
#
print "\t----\t getunit Ex 1 \t----"
tu = qa.quantity(-1.3, 'Jy')         # make quantity
print tu
#{'value': -1.3, 'unit': 'Jy'}
print qa.getvalue(tu)
#-1.3
print qa.getunit(tu)
#Jy
#
"""
```

quanta.canonical.html

quanta.canonical - Function

1.5.1 get canonical value of quantity

Description

canonical (with alias canon) gets the canonical value of a quantity

Arguments

Inputs		
v	value to convert	
	allowed:	variant
	Default:	1.0

Returns

record

Example

```
"""
#
print "\t----\t canonical Ex 1 \t----"
print qa.canonical('1Jy') # canonical value of a string
#{'value': 1e-26, 'unit': 'kg.s-2'}
print qa.canon(qa.quantity('1Jy')) # canonical value of a unit
#{'value': 1e-26, 'unit': 'kg.s-2'}
#
"""
```

quanta.canon.html

quanta.canon - Function

1.5.1 get canonical value of quantity

Description

canon gets the canonical value of a quantity

Arguments

Inputs		
v	value to convert	
	allowed:	variant
	Default:	

Returns

record

Example

```
"""
#
print "\t----\t canon Ex 1 \t----"
print qa.canon('1Jy')         # canonical value of a string
#{'value': 1e-26, 'unit': 'kg.s-2'}
print qa.canonical(qa.quantity('1Jy')) # canonical value of a unit
#{'value': 1e-26, 'unit': 'kg.s-2'}
#
"""
```

quanta.convert.html

quanta.convert - Function

1.5.1 convert a quantity to another unit

Description

convert converts a quantity to another unit. If no output unit given, conversion is to canonical units

Arguments

Inputs		
v	quantity to convert	
	allowed:	variant
	Default:	
outunit	unit to convert to	
	allowed:	variant
	Default:	

Returns

record

Example

```
"""
#
print "\t----\t convert Ex 1 \t----"
tu = qa.quantity('5Mm/s') # specify a quantity
print tu
#{'value': 5.0, 'unit': 'Mm/s'}
print qa.convert(tu, 'pc/a') # convert it to parsec per year
#{'value': 0.0051135608266237404, 'unit': 'pc/a'}
print qa.convert(tu) # convert to canonical units
#{'value': 5000000.0, 'unit': 'm.s-1'}
#
```

"""

---

quanta.define.html

quanta.define - Function

1.5.1 define a new unit name

Description

define defines the name and value of a user defined unit

Arguments

Inputs		
name	name of unit to define	
	allowed:	string
	Default:	
v	quantity value of new unit	
	allowed:	variant
	Default:	1

Returns
bool

Example

```
"""
#
print "\t----\t define Ex 1 \t----"
print qa.define('JY','1Jy') # your misspelling
#True
print qa.define('VLAunit', '0.898 JY') # a special unit using it
#True
print qa.quantity('5 VLAunit')  # check its use
#{'value': 5.0, 'unit': 'VLAunit'}
print qa.convert('5 VLAunit','Jy')
#{'value': 4.4900000000000002, 'unit': 'Jy'}
#
```

"""

---

quanta.map.html

quanta.map - Function

1.5.1 list known unit names and constants

Description

map lists the known mapping of units and constants. It has a single argument, which can be a coded string (no-case, minimax match):

all all of the following units (not constants): also the default

Prefix known decimal prefixes

SI known SI units

Customary a set of customary units known to programs

User units defined by the user

Constants known constants (note: only 'const', 'Const', 'constants' and 'Constants' recognised).

Arguments

Inputs		
v	type of information to list - coded string	
	allowed:	string
	Default:	all

Returns

string

Example

```
"""
#
print "\t----\t map Ex 1 \t----"
```

```
print qa.map('pre') # list decimal prefixes
#        == Prefix ==== 20 ====
#        E         (exa)                        1e+18
#        G         (giga)                       1000000000
#        M         (mega)                       1000000
#        P         (peta)                       1e+15
#        T         (tera)                       1e+12
#        Y         (yotta)                      1e+24
#        Z         (zetta)                      1e+21
#        a         (atto)                       1e-18
#        c         (centi)                      0.01
#        d         (deci)                       0.1
#        da        (deka)                       10
#        f         (femto)                      1e-15
#        h         (hecto)                      100
#        k         (kilo)                       1000
#        m         (milli)                      0.001
#        n         (nano)                       1e-09
#        p         (pico)                       1e-12
#        u         (micro)                      1e-06
#        y         (yocto)                      1e-24
#        z         (zepto)                      1e-21
print qa.map('Constants') # list known constants
#        == Constants ====
#        pi    3.14..                    3.14159
#        ee    2.71..                    2.71828
#        c     light vel.                2.99792e+08 m/s
#        G     grav. const               6.67259e-11 N.m2/kg2
#        h     Planck const              6.62608e-34 J.s
#        HI    HI line                   1420.41 MHz
#        R     gas const                 8.31451 J/K/mol
#        NA    Avogadro #                6.02214e+23 mol-1
#        e     electron charge           1.60218e-19 C
#        mp    proton mass               1.67262e-27 kg
#        mp_me mp/me                     1836.15
#        mu0   permeability vac.         1.25664e-06 H/m
#        eps0  permittivity vac.         1.60218e-19 C
#        k     Boltzmann const           1.38066e-23 J/K
#        F     Faraday const             96485.3 C/mol
#        me    electron mass             9.10939e-31 kg
#        re    electron radius           2.8179e-15 m
#        a0    Bohr's radius             5.2918e-11 m
#        R0    solar radius              6.9599e+08 m
#        k2    IAU grav. const^2         0.000295912 AU3/d2/S0
#
"""
```

quanta.maprec.html

quanta.maprec - Function

1.5.1 create record containing list of known unit names and constants

Description

maprec returns a record with the known mapping of units and constants. It has a single argument, which can be a coded string (no-case, minimax match):

all all of the following units (not constants): also the default

Prefix known decimal prefixes

SI known SI units

Customary a set of customary units known to programs

User units defined by the user

Arguments

Inputs		
v	type of information to list - coded string	
	allowed:	string
	Default:	all

Returns

record

Example

```
"""
#
print "\t----\t maprec Ex 1 \t----"
p = qa.maprec('pre') # list decimal prefixes
print p['Prefix_G']
```

```
#        G         (giga)               1000000000
s = qa.maprec('SI')         # list SI units
print s['SI_Jy']
#Jy        (jansky)                    1e-26 kg.s-2
#
"""
```

---

quanta.fits.html

quanta.fits - Function

1.5.1 define some FITS units

Description

fits defines some unit names used in reading and writing FITS files.

Arguments

Inputs

Returns
bool

Example

```
"""
#
print "\t----\t fits Ex 1 \t----"
print qa.fits()
#True
print qa.map('user')
#        == User ====
#        BEAM      (dimensionless beam)         1 _
#        DAYS      (day)                        86400 s
#        DEG       (degree)                     0.0174532925199 rad
#        DEGREES   (degree)                     0.0174532925199 rad
#        HZ        (hertz)                      1 s-1
#        JY        (jansky)                     1e-26 kg.s-2
#        KELVIN    (kelvin)                     1 K
#        KELVINS   (kelvin)                     1 K
#        KM        (km)                         1000 m
#        M         (meter)                      1 m
```

```
#        METERS    (meter)                    1 m
#        PASCAL    (pascal)                   1 m-1.kg.s-2
#        PIXEL     (dimensionless pixel)      1 _
#        S         (second)                   1 s
#        SEC       (second)                   1 s
#        SECONDS   (second)                   1 s
#        VOLTS     (volt)                     1 m2.kg.s-3.A-1
#        YEAR      (year)                     31557600 s
#        YEARS     (year)                     31557600 s
#
"""
```

---

quanta.angle.html

quanta.angle - Function

1.5.1 show an angle as a formatted string

Description

angle converts an angle quantity to a formatted string. The formatting information is a precision (0 is default, 6 includes +-ddd.mm.ss) and a string array of codes (no-case, minimax match): Codes include:

clean delete leading/trailing superfluous separators

no_d do not show degrees part

no_dm do not show degrees and minutes part

dig2 show only 2 digits of degrees in angle format

time show as time (hh:mm:ss.ttt) rather than as angle

If a multi-dimensional value is given for the value v, the returned value is a string vector of a length equal to last dimension. Each string has a number of fields equal to the number of elements in all earlier dimensions. If the *showform* is T, each vector element is surrounded by a pair of square brackets if there is more than one entry, and fields are separated by a ','.

Arguments

Inputs		
v	angle quantity value to output	
	allowed:	variant
	Default:	
prec	number of digits shown	
	allowed:	int
	Default:	0
form	formatting information in coded string array	
	allowed:	stringArray
	Default:	
showform	show square brackets and separating ,	
	allowed:	bool
	Default:	false

Returns
stringArray

Example

```
"""
#
print "\t----\t angle Ex 1 \t----"
tu = qa.quantity('5.7.12.345678') # define an angle
print tu
#{'value': 5.1200960216666669, 'unit': 'deg'}
print qa.angle(tu)     # default output
#+005.07.12
print qa.angle(tu, prec=7) # 7 digits
#+005.07.12.3
print qa.angle(tu, prec=4) # 4 digits
#+005.07.
print qa.angle(tu, form=["tim","no_d"]) # as time, no hours shown
#:20:29
#
"""
```

---

quanta.time.html

quanta.time - Function

1.5.1 show a time (or date) as a formatted string

Description

time converts a time quantity to a formatted string. The formatting information is a precision (0 is default, 6 includes hh.mm.ss) and a string array of codes (no-case, minimax match): Codes include:

clean delete leading/trailing superfluous separators

no_d do not show hours part

no_dm do not show hours and minutes part

ymd include a date as yyyy/mm/dd (date is by default not shown)

dmy include a date as ddMMMyyyy (date is by default not shown)

mjd include a date as Modified Julian Day (date is by default not shown)

fits include a date and show time in FITS format: le from OS

angle show in angle (dd.mm.ss.ttt) rather than time format

day prefix day-of-week to output

local show local time rather than UTC (add timezone offset)

no_time suppress printing of time part

If a multi-dimensional value is given for the value v, the returned value is a string vector of a length equal to last dimension. Each string has a number of fields equal to the number of elements in all earlier dimensions. If the *showform* is T, each vector element is surrounded by a pair of square brackets if there is more than one entry, and fields are separated by a ','.

Arguments

Inputs		
v	time quantity value to output	
	allowed:	variant
	Default:	
prec	number of digits shown	
	allowed:	int
	Default:	0
form	formatting information in coded string array	
	allowed:	stringArray
	Default:	
showform	show square brackets and separating ,	
	allowed:	bool
	Default:	false

Returns
stringArray

Example

```
"""
#
print "\t----\t time Ex 1 \t----"
tu = qa.quantity('today') # a time
print tu
#{'value': 54175.708981504627, 'unit': 'd'}
print qa.time(tu) # default format
#17:00:56
print qa.time(tu,form="dmy")   # show date
#16-Mar-2007/17:00:56
print qa.time(tu,form=["ymd","day"]) # and day
#Fri-2007/03/16/17:00:56
print qa.time(tu,form="fits")           # FITS format
#2007-03-16T17:00:56
print qa.time(tu,form=["fits","local"]) # local FITS format
#2007-03-16T10:00:56-07:00
print qa.time(tu,form=["ymd","local"])  # local time
#2007/03/16/10:00:56
#
"""
```

quanta.add.html

quanta.add - Function

1.5.1 add quantities

Description

add adds two quantities

Arguments

Inputs		
v	value	
	allowed:	variant
	Default:	
a	value	
	allowed:	variant
	Default:	0

Returns

record

Example

```
"""
#
print "\t----\t add Ex 1 \t----"
print qa.add('5m', '2yd')
#{'value': 6.8288000000000002, 'unit': 'm'}
#
"""
```

quanta.sub.html

quanta.sub - Function

1.5.1 subtract quantities

Description

sub subtracts two quantities

Arguments

Inputs		
v	value	
	allowed:	variant
	Default:	
a	value	
	allowed:	variant
	Default:	variant 0

Returns
record

Example

```
"""
#
print "\t----\t sub Ex 1 \t----"
print qa.sub('5m', '2yd')
#{'value': 3.1712000000000002, 'unit': 'm'}
#
"""
```

quanta.mul.html

quanta.mul - Function

1.5.1 multiply quantities

Description

mul multiplies two quantities

Arguments

Inputs		
v	value	
	allowed:	variant
	Default:	
a	value	
	allowed:	variant
	Default:	1

Returns

record

Example

```
"""
#
print "\t----\t mul Ex 1 \t----"
print qa.mul('5m', '3s')
#{'value': 15.0, 'unit': 'm.s'}
#
"""
```

quanta.div.html

quanta.div - Function

1.5.1 divides quantities

Description

div divides two quantities

Arguments

Inputs		
v	value	
	allowed:	variant
	Default:	
a	value	
	allowed:	variant
	Default:	1

Returns
record

Example

```
"""
#
print "\t----\t div Ex 1 \t----"
print qa.div('5m', '3s')
#{'value': 1.6666666666666667, 'unit': 'm/(s)'}
#
"""
```

quanta.neg.html

quanta.neg - Function

1.5.1 negate quantities

Description

neg negates a quantity

Arguments

Inputs		
v	value	
	allowed:	variant
	Default:	1

Returns

record

Example

```
"""
#
print "\t----\t neg Ex 1 \t----"
print qa.neg('5m')
#{'value': -5.0, 'unit': 'm'}
#
"""
```

quanta.norm.html

quanta.norm - Function

1.5.1 normalise angle

Description

norm normalise angles in interval of 2π radians. The default interval is from -0.5 to +0.5 of a full interval (i.e. from -180 to +180 degrees). The lower end of the interval can be set as a fraction of 2π

Arguments

Inputs		
v	angle quantity	
	allowed:	variant
	Default:	
a	lower interval boundary	
	allowed:	double
	Default:	-0.5

Returns

record

Example

```
"""
#
print "\t----\t norm Ex 1 \t----"
print qa.norm('713deg') #default normalisation
#{'value': -6.9999999999999716, 'unit': 'deg'}
print qa.norm('713deg', -2.5)  # normalise to interval -900 - -540 deg
#{'value': -727.0, 'unit': 'deg'}
#
"""
```

quanta.le.html

quanta.le - Function

1.5.1 compare quantities

Description

le compares two quantities for less than or equal.

Arguments

Inputs		
v	value	
	allowed:	any
	Default:	variant
a	value	
	allowed:	any
	Default:	variant 0

Returns

bool

Example

```
"""
#
print "\t----\t le Ex 1 \t----"
print qa.le('5m', '2yd')
#False
#
"""
```

quanta.lt.html

quanta.lt - Function

1.5.1 compare quantities

Description

lt compares two quantities for less than.

Arguments

Inputs		
v	value	
	allowed:	any
	Default:	variant
a	value	
	allowed:	any
	Default:	variant 0

Returns

bool

Example

```
"""
#
print "\t----\t lt Ex 1 \t----"
print qa.lt('5m', '2yd')
#False
#
"""
```

quanta.eq.html

quanta.eq - Function

1.5.1 compare quantities

Description

eq compares two quantities for equality.

Arguments

Inputs		
v	value	
	allowed:	any
	Default:	variant
a	value	
	allowed:	any
	Default:	variant 0

Returns

bool

Example

```
"""
#
print "\t----\t eq Ex 1 \t----"
print qa.eq('5m', '2yd')
#False
#
"""
```

quanta.ne.html

quanta.ne - Function

1.5.1 compare quantities

Description

ne compares two quantities for non equality.

Arguments

Inputs		
v	value	
	allowed:	any
	Default:	variant
a	value	
	allowed:	any
	Default:	variant 0

Returns

bool

Example

```
"""
#
print "\t----\t ne Ex 1 \t----"
print qa.ne('5m', '2yd')
#True
#
"""
```

quanta.gt.html

quanta.gt - Function

1.5.1 compare quantities

Description

gt compares two quantities for greater than.

Arguments

Inputs		
v	value	
	allowed:	any
	Default:	variant
a	value	
	allowed:	any
	Default:	variant 0

Returns

bool

Example

```
"""
#
print "\t----\t gt Ex 1 \t----"
print qa.gt('5m', '2yd')
#True
#
"""
```

quanta.ge.html

quanta.ge - Function

1.5.1 compare quantities

Description

ge compares two quantities for greater than or equal.

Arguments

Inputs		
v	value	
	allowed:	any
	Default:	variant
a	value	
	allowed:	any
	Default:	variant 0

Returns

bool

Example

```
"""
#
print "\t----\t ge Ex 1 \t----"
print qa.ge('5m', '2yd')
#True
#
"""
```

quanta.sin.html

quanta.sin - Function

1.5.1 sine of quantity

Description

sin gives sine of angle quantity

Arguments

Inputs		
v	angle quantity	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t sin Ex 1 \t----"
print qa.sin('7deg')
#{'value': 0.12186934340514748, 'unit': ''}
#
"""
```

---

quanta.cos.html

quanta.cos - Function

1.5.1 cosine of quantity

Description

cos gives cosine of angle quantity

Arguments

Inputs		
v	angle quantity	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t cos Ex 1 \t----"
print qa.cos('7deg')
#{'value': 0.99254615164132198, 'unit': ''}
#
"""
```

quanta.tan.html

quanta.tan - Function

1.5.1 tangent of quantity

Description

tan gives tangent of angle quantity

Arguments

Inputs		
v	angle quantity	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t tan Ex 1 \t----"
print qa.tan('7deg')
#{'value': 0.1227845609029046, 'unit': ''}
#
"""
```

---

quanta.asin.html

quanta.asin - Function

1.5.1 arcsine of quantity

Description

asin gives arcsine of non-dimensioned quantity

Arguments

Inputs		
v	non-dimensioned quantity	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t asin Ex 1 \t----"
print qa.convert(qa.asin(qa.sin('7deg')), 'deg')
#{'value': 7.0, 'unit': 'deg'}
#
"""
```

quanta.acos.html

quanta.acos - Function

1.5.1 arccosine of quantity

Description

acos gives arccosine of non-dimensioned quantity

Arguments

Inputs		
v	non-dimensioned quantity	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t acos Ex 1 \t----"
print qa.convert(qa.acos(qa.cos('7deg')), 'deg')
#{'value': 7.0000000000000249, 'unit': 'deg'}
#
"""
```

quanta.atan.html

quanta.atan - Function

1.5.1 arctangent of quantity

Description

atan gives arctangent of non-dimensioned quantity

Arguments

Inputs		
v	non-dimensioned quantity	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t atan Ex 1 \t----"
print qa.convert(qa.atan(qa.tan('7deg')), 'deg')
#{'value': 7.0, 'unit': 'deg'}
#
"""
```

quanta.atan2.html

quanta.atan2 - Function

1.5.1 arctangent of two quantity

Description

atan gives arctangent of two non-dimensioned quantity

Arguments

Inputs		
v	non-dimensioned quantity	
	allowed:	any
	Default:	variant
a	non-dimensioned quantity	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t atan2 Ex 1 \t----"
print qa.convert(qa.atan2(qa.sin('7deg'), qa.cos('7deg')), 'deg')
#{'value': 7.0, 'unit': 'deg'}
#
"""
```

quanta.abs.html

quanta.abs - Function

1.5.1 absolute value of quantity

Description

abs gives absolute value of quantity

Arguments

Inputs		
v	value	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t abs Ex 1 \t----"
print qa.abs('-5km/s')
#{'value': 5.0, 'unit': 'km/s'}
#
"""
```

quanta.ceil.html

quanta.ceil - Function

1.5.1 ceil value of quantity

Description

ceil gives ceiling value of quantity

Arguments

Inputs		
v	value	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t ceil Ex 1 \t----"
print qa.ceil('5.1AU')
#{'value': 6.0, 'unit': 'AU'}
#
"""
```

quanta.floor.html

quanta.floor - Function

1.5.1 floor value of quantity

Description

floor gives flooring value of quantity

Arguments

Inputs		
v	value	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t floor Ex 1 \t----"
print qa.floor('-5.1AU')
#{'value': -6.0, 'unit': 'AU'}
#
"""
```

quanta.log.html

quanta.log - Function

1.5.1 logarithm of quantity

Description

log gives natural logarithm of dimensionless quantity

Arguments

Inputs		
v	dimensionless quantity	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t log Ex 1 \t----"
print qa.log('2')
#{'value': 0.69314718055994529, 'unit': ''}
#
"""
```

quanta.log10.html

quanta.log10 - Function

1.5.1 logarithm of quantity

Description

log10 gives logarithm of dimensionless quantity

Arguments

Inputs		
v	dimensionless quantity	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t log10 Ex 1 \t----"
print qa.log10('2')
#{'value': 0.3010299956639812, 'unit': ''}
#
"""
```

quanta.exp.html

quanta.exp - Function

1.5.1 exponential of quantity

Description

exp gives exponential value of dimensionless quantity

Arguments

Inputs		
v	dimensionless quantity	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t exp Ex 1 \t----"
print qa.exp('2')
#{'value': 7.3890560989306504, 'unit': ''}
try:
  print qa.exp('2m')
except Exception, e:
  print "Caught an expected exception", e
#Caught an expected exception Quantum::exp illegal unit type 'm'
#
"""
```

quanta.sqrt.html

quanta.sqrt - Function

1.5.1 square root of quantity

Description

sqrt gives square root of quantity with only even powered dimensions

Arguments

Inputs		
v	dimensionless	quantity
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t sqrt Ex 1 \t----"
print qa.sqrt('2m2')
#{'value': 1.4142135623730951, 'unit': 'm'}
try:
  print qa.sqrt('2s')
except Exception, e:
  print "Caught an expected exception", e
#Caught an expected exception UnitVal::UnitVal Illegal unit dimensions for root
#
"""
```

quanta.compare.html

quanta.compare - Function

1.5.1 compare dimensionality of units

Description

compare compares the dimensionality of units of two qauntities

Arguments

Inputs		
v	value	
	allowed:	any
	Default:	variant
a	value	
	allowed:	any
	Default:	variant

Returns

bool

Example

```
"""
#
print "\t----\t compare Ex 1 \t----"
print qa.compare('5yd/a', '6m/s')   # equal dimensions
#True
print qa.compare('5yd', '5s') # unequal dimensions
#False
#
"""
```

quanta.check.html

quanta.check - Function

1.5.1 check for proper unit string

Description

check checks if the argument has a properly defined unit string

Arguments

Inputs		
v	value	
	allowed:	string
	Default:	

Returns

bool

Example

```
"""
#
print "\t----\t check Ex 1 \t----"
print qa.check('5AE/Jy.pc5/s')
#True
print qa.check('7MYs')
#False
#
"""
```

quanta.checkfreq.html

quanta.checkfreq - Function

1.5.1 check for proper frequency unit

Description

checkfreq checks if the argument has a properly defined frequency interpretable unit string

Arguments

Inputs		
cm	value	
	allowed:	any
	Default:	variant

Returns

bool

Example

```
"""
#
print "\t----\t checkfreq Ex 1 \t----"
print qa.checkfreq('5GHz')
#True
print qa.checkfreq('5cm')
#True
print qa.checkfreq('5cm/s2')
#False
#
"""
```

quanta.pow.html

quanta.pow - Function

1.5.1 raise quantity to power

Description

pow raises a quantity to an integer power

Arguments

Inputs		
v	value	
	allowed:	any
	Default:	variant
a	power	
	allowed:	int
	Default:	1

Returns

record

Example

```
"""
#
print "\t----\t pow Ex 1 \t----"
print qa.pow('7.2km/s', -3)
#{'value': 0.0026791838134430724, 'unit': '(km/s)-3'}
#
"""
```

quanta.constants.html

quanta.constants - Function

1.5.1 get a constant

Description

constants gets a named constant quantity. Names (no-case, minimax) are: pi 3.14.. 3.14159 ee 2.71.. 2.71828 c light vel. 2.99792e+08 m/s G grav. const 6.67259e-11 N.m2/kg2 h Planck const 6.62608e-34 J.s HI HI line 1420.41 MHz R gas const 8.31451 J/K/mol NA Avogadro number 6.02214e+23 mol-1 e electron charge 1.60218e-19 C mp proton mass 1.67262e-27 kg mp_me mp/me 1836.15 mu0 permeability vac. 1.25664e-06 H/m eps0 permittivity vac. 1.60218e-19 C k Boltzmann const 1.38066e-23 J/K F Faraday const 96485.3 C/mol me electron mass 9.10939e-31 kg re electron radius 2.8179e-15 m a0 Bohr's radius 5.2918e-11 m R0 solar radius 6.9599e+08 m k2 IAU grav. const$\hat{2}$ 0.000295912 AU3/d2/S0

Arguments

Inputs		
v	name	
	allowed:	string
	Default:	pi

Returns

record

Example

```
"""
#
print "\t----\t constants Ex 1 \t----"
print qa.constants()
#{'unit': '', 'value': 3.1415926535897931}
#
```

"""

---

quanta.isangle.html

quanta.isangle - Function

1.5.1 check if valid angle or time quantity

Description

isangle checks if the argument is a valid angle/time quantity.

Arguments

Inputs		
v	angle/time quantity	
	allowed:	any
	Default:	variant

Returns

bool

Example

```
"""
#
print "\t----\t isangle Ex 1 \t----"
print qa.isangle(qa.constants('pi'))
#False
#
"""
```

---

quanta.totime.html

quanta.totime - Function

1.5.1 convert an angle (or a time) to a time

Description

totime converts an angle quantity (or a time) to a time quantity

Arguments

Inputs		
v	angle/time quantity	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t totime Ex 1 \t----"
print qa.totime('2d5m')
#{'value': 0.0057870370370370376, 'unit': 'd'}
#
"""
```

---

quanta.toangle.html

quanta.toangle - Function

1.5.1 convert a time (or an angle) to an angle

Description

toangle converts a time quantity (or an angle) to an angle quantity

Arguments

Inputs		
v	angle/time quantity	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t toangle Ex 1 \t----"
print qa.toangle('5h30m12.6')
#{'value': 82.552499999999995, 'unit': 'deg'}
#
"""
```

---

quanta.splitdate.html

quanta.splitdate - Function

1.5.1 split a date/time into a record

Description

splitdate splits a date/time quantity into a record with constituent fields like year, yearday, month etc. All fields will be integer (to enable use as index and easy personal formatting), with the exception of the *s* field which is a double float. See the example for the fields returned.

Arguments

Inputs		
v	angle/time quantity	
	allowed:	any
	Default:	variant

Returns

record

Example

```
"""
#
print "\t----\t splitdate Ex 1 \t----"
print qa.splitdate('today')

#{'mjd': 54175.752367291658, 'week': 11, 'usec': 533999, 'hour': 18,
# 'min': 3, 'yearday': 75, 'msec': 533, 'month': 3, 's':
# 24.533999226987362, 'sec': 24, 'weekday': 5, 'year': 2007, 'monthday':
# 16} print qa.splitdate('183.33333333deg')
#{'mjd': 0.50925925925000004, 'week': 46, 'usec': 999999, 'hour': 12,
# 'min': 13, 'yearday': 321, 'msec': 999, 'month': 11, 's':
# 19.999999200003487, 'sec': 19, 'weekday': 3, 'year': 1858,
```

```
# 'monthday': 17}
#
"""
```

---

quanta.tos.html

quanta.tos - Function

1.5.1 convert quantity to string

Description

tos converts a quantity to a string with the precision defined with the *setformat('prec')* (which defaults to 9). If the optional *prec* argument is set to an integer value greater than 1, that precision is used in the conversion

Arguments

Inputs		
v	value	
	allowed:	any
	Default:	variant
prec	convert precision of value	
	allowed:	int
	Default:	9

Returns
string

Example

```
"""
#
print "\t----\t tos Ex 1 \t----"
a = qa.quantity('2.56 yd/s')
print a
#{'value': 2.5600000000000001, 'unit': 'yd/s'}
print qa.tos(a)
#2.560000000yd/s
a=qa.quantity(1./7, 'km/s')
print qa.tos(a)
```

```
#0.142857143km/s
print qa.tos(a,2)
#0.14km/s
print qa.tos(a,20)
#0.14285714285714284921km/s
print qa.tos(a)
#0.142857143km/s
#
"""
```

---

quanta.type.html

quanta.type - Function

1.5.1 type of tool

Description

type will return the tool name.

Arguments

Inputs

Returns

string

Example

```
"""
#
print "\t----\t type Ex 1 \t----"
print qa.type()
#quanta
#
"""
```

---

quanta.done.html

quanta.done - Function

1.5.1 Free resources used by tool. Current implementation ignores input parameter, does nothing and returns true

Description

Currently, this method is an NOP.

Arguments

Inputs		
kill	force kill of the default tool (ignored)	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
"""
#
print "\t----\t done Ex 1 \t----"
print qa.done()
#True
print qa.done()
#True
print qa.done(kill=T)
#True
#
"""
```

quanta.unit.html

quanta.unit - Function

1.5.1 quantity from value v and unit string

Description

unit makes a quantity from a string, or from a value and a string. Note that unit is a synonym for quantity (see example there).

Arguments

Inputs		
v		
	allowed:	any
	Default:	variant
unitname		
	allowed:	string
	Default:	

Returns

record

---

quanta.isquantity.html

quanta.isquantity - Function

1.5.1 Check if quantity

Description

Checks if the operand is a correct quantity

Arguments

Inputs		
v	value to be tested	
	allowed:	any
	Default:	variant

Returns

bool

Example

```
"""
#
print "\t----\t isQuantity Ex 1 \t----"
a = qa.quantity("5Jy")               # make a quantity
print a
#{'value': 5.0, 'unit': 'Jy'}
print qa.isquantity(a)                  # is it one?
#True
print qa.isquantity("5Jy")              # and this string?
#True
#
"""
```

quanta.setformat.html

quanta.setformat - Function

1.5.1 set format for output of numbers. (NOT IMPLEMENTED YET!)

Arguments

Inputs		
t	type -coded string indicating which format parameter to set	
	allowed:	string
	Default:	
v	format parameter value - numeric or coded string, depending on format type to be set	
	allowed:	string
	Default:	F

Returns
bool

---

quanta.getformat.html

quanta.getformat - Function

1.5.1 get current output format (NOT IMPLEMENTED YET!)

Description

getformat returns the current format value set for the different format possibilities. See the setformat function for the different format type descriptions. The known types are:
prec, aprec, tprec, long, lat, len, dtime, elev, auto, vel, freq, dop, unit.

Arguments

Inputs		
t	type - coded string	
	allowed:	string
	Default:	

Returns

string

Example

```
"""
#
print "\t----\t getformat Ex 1 \t----"
print qa.getformat('prec')
#6
#setformat is NOT IMPLEMENTED YET!
#qa.setformat('prec', 12) # set precision to 12 significant digits
#T
#print qa.getformat('prec')
#12
print qa.getformat('long')
#hms
```

```
#
"""
```

---

quanta.formxxx.html

quanta.formxxx - Function

1.5.1 Format a quantity using given format, allowed are hms, dms, deg, rad, +deg.

Description

form.xxx (xxx can be lat, long, len, vel, freq, dtime, unit) will format the input into a string using the global format information set by setformat().

Arguments

Inputs		
v	value to be converted	
	allowed:	any
	Default:	variant
format	xxx can be hms, dms, deg, rad or +deg	
	allowed:	string
	Default:	dms
prec	digits in fractional part of output string for dms,hms	
	allowed:	int
	Default:	2

Returns

string

Example

```
"""
#
print "\t----\t formxxx Ex 1 \t----"
#qa.setformat('freq','cm')
#T
#qa.formxxx('freq',qa.quantity('5GHz'))
#form_xxx NOT IMPLEMENTED YET!
```

```
#5.99584916 cm
print "Last example, exiting! ..."
exit()
#
"""
```

---

spectralline-Module.html

1.6 spectralline - Module

Spectral line search from Splatalogue

Description

This module contains functionality to access spectral-line Splatalogue line list.
The available tools in this module are

- spectralline - In order for the spectralline tool to be able to search Splatalogue a splatalogue line list, an ascii version of a line list must be exported using the splatalogue web interface and then converted to a CASA table. The method splattotable does this conversion.

Images

SynthesisPackage.html

Chapter 2

Package Synthesis

The synthesis package contains modules needed for processing synthesis data. calibrater-Module.html

2.1 calibrater - Module

Module for synthesis calibration
include calibrater.g

Description The `calibrater` module provides synthesis calibration capabilities within CASA. The primary purpose of this module is to solve for calibration components, and to optionally apply these corrections to the observed data. The calibration module is designed to be used in conjunction with the imager module which provides support for synthesis imaging.

The calibration model adopted by `calibrater` is that of the Hamaker-Bregman-Sault measurement equation for synthesis radio telescopes (see CASANote 189). This represents calibration corrections as matrices acting on 4-vectors representing the four possible correlations measured by an interferometer in full polarization. The calibration matrices cover a diversity of instrumental effects, including: parallactic angle and feed configuration (P,C), atmospheric phase (T), electronic gain (G), bandpass (B), instrumental polarization (D), baseline-based (correlator) corrections (M, MF), and baseline-based fringe-fitting (K).

The calibration data are stored in CASAtables and can be directly examined, manipulated or edited in the Glish command line interpreter (CLI) via the table tool. The calibration tables may be interpolated when applied to the observed uv-data.

The solver allows the calibration components to be determined over different time intervals, thus allowing, as an example, the solution for atmospheric phase effects (T) over a much shorter interval than electronic gain terms (G). This also allows polarization self-calibration for time variable instrumental polarization corrections.

The measurement equation is designed to model calibration effects for a generic radio telescope and the calibration and synthesis modules are, in general, not instrument specific.

Each `calibrater` tool created acts on a specified Measurement Set (MS), containing the observed uv-data. The Measurement Set format is described in (see CASANote 191). The interaction of the `calibrater` tool with specific MS data columns is important. The observed data, as recorded by the instrument, are stored in the **DATA** column of the MS, and are referred to as the observed data. If calibration corrections are applied by `calibrater`, the resulting calibrated data are stored in a separate **CORRECTED_DATA** column in the MS. These columns can be selected when imaging the data using the imager tool. A further MS column is used by `calibrater`, namely the **MODEL_DATA** column. The difference between the model data and corrected data columns is used to form χ^2, when solving for individual calibration components. It is important to set the **MODEL_DATA** column before using `calibrater` to solve for calibration. This can be done using the `imager` functions setjy or ft.

The capabilities of the `calibrater` module are made available by including the associated Glish initialization script for the module, as:

```
- include 'calibrater.g'
T
```

where a hyphen precedes user input. The Glish response is indicated without the prompt.

A `calibrater` tool is created and attached to a specified measurement set as indicated in the following example:

```
- c:=calibrater('3C273XC1.MS');
T
```

A variety of functions can be invoked for any given `calibrater` tool. These functions fall broadly into two categories: i) functions which set parameters to be used by the calibrater; and ii) the execution of explicit calibration procedures such as solving for, or applying calibration corrections.

Option (i) may equivalently be viewed as setting the state of the `calibrater` tool. These functions are named with the prefix `set`, such as in `setdata` and `setapply`. When created, the `calibrater` tool sets default internal information for each of the calibration components (measurement equation correction matrices). This information is modified using the `setapply` and `setsolve` functions as shown in the following example:

```
#
# Set the solution interval for the electronic gain matrix (G) to
# 300 seconds, specify input and output calibration table names,
# and enable this component for phase and amplitude solution.
# Use antenna number 3 as the reference for the solutions.
#
```

```
- c.setapply ("G", 0.0, "gcal_in", "");
T
- c.setsolve ("G", 300, F, 3, "gcal_out", F);
T
```

Once the state of the `calibrater` tool has been set, explicit calibration functions, as outlined in option (ii) above, are executed as follows:

```
#
# Solve for the selected calibration components
#
- c.solve()
T
#
# Apply the calibration components to the measurement set data
#
- c.correct()
T
```

Example The following example illustrates the quickest way to perform simple self-calibration, starting from an input FITS file in the local area. The `imager` module should be consulted for detailed information on the imaging functions.

```
#
# Include the synthesis scripts
#
include 'imager.g';
include 'calibrater.g';
include 'ms.g';
#
# Construct a measurement set from the input FITS file
#
m:=fitstoms (msfile='3C273XC1.MS', fitsfile='3C273XC1.FITS');
m.close();
m.done();
#
# Create an imager tool
#
sk:=imager('3C273XC1.MS');
#
# Set image parameters
#
sk.setimage (nx=256, ny=256, cellx='0.7arcsec', celly='0.7arcsec');
#
# Make a dirty image and deconvolve using CLEAN
```

```
#
sk.image ('observed', image='3C273XC1.dirty');
sk.clean (niter=1000, threshold='3mJy', model='3C273XC1.clean.model');
#
# Fourier transform the model to the uv-plane
#
sk.ft (model='3C273XC1.clean.model');
#
# Close the imager tool
#
sk.close();
sk.done();
#
# Create a calibrater tool
#
c:=calibrater('3C273XC1.MS');
#
# Select solution for electronic gain (G) and atmospheric phase (T)
#
c.setsolve ("G", 300.0, F, 3, "gcal_out", F);
c.setsolve ("T", 30.0, T, 3, "tcal_out", F);
#
# Solve for the selected G and T components
#
c.solve();
#
# Close the calibrater tool
#
c.close();
```

calibrater-Tool.html

2.1.1 calibrater - Tool

Synthesis calibration (self- and cross-)
Requires:

Synopsis

Description

The `calibrater` tool (cb) provides for synthesis calibration operations within CASA.

Methods

calibrater	Construct a calibrater tool
open	Attach MeasurementSet to the calibrater tool
selectvis	Set the data selection for subsequent processing
setmodel	Set the sky model used to compute the model visibilities
setptmodel	Set the point source model Stokes parameters to be used to compute the model visibilit
setapply	Arrange to apply calibration
setcallib	Arrange to apply calibration via a Cal Library
validatecallib	Validate a Cal Library record
setsolve	Arrange to solve for calibration
setsolvegainspline	Specialization of setsolve for cubic spline G (time-dependent gain) solving
setsolvebandpoly	Specialization of setsolve for polynomial B (bandpass) solving
state	Request the apply/solve state of the calibrater tool
reset	Reset the selected apply and/or solve components
initcalset	Re-initialize the calibration scratch columns.
delmod	Delete model data representations in the MS.
solve	Solve for the selected calibration components
correct	Apply calibration information
corrupt	Corrupt model with calibration tables
initweights	Initialize MS weight info in various ways.
fluxscale	Bootstrap the flux density scale from standard calibrators
accumulate	Accumulate incremental calibration solutions into a cumulative calibration table
activityrec	Returns a record containing properties of recent activity
specifycal	Externally specify calibration of various types
smooth	Produce a smoothed calibration table
listcal	List the contents of a calibration table
posangcal	Apply position angle calibration to an existing cal table

linpolcor	Correct the gain table for linear polarization of the calibrator
plotcal	Plot a calibration table
modelfit	Model fitting
updatecaltable	Caltable modernizer.
close	Close the calibrater tool
done	Destroy the calibrater tool

calibrater.calibrater.html

calibrater.calibrater - Function

2.1.1 Construct a calibrater tool

Description

Create a `calibrater` tool. The casapy environment provides a standard calibrater tool for general use (cb), but additional calibrater tools may be created if needed. Calibrater tools created in this way are independent of the standard calibrater tool.

Arguments

Returns
calibrater

Example

```
cb2=calibrater.create()
```

---

calibrater.open.html

calibrater.open - Function

2.1.1 Attach MeasurementSet to the calibrater tool

Description

Attaches a MeasurementSet to the `calibrater` tool for further processing with other methods.

Arguments

Inputs		
filename	MeasurementSet file name. No default	
	allowed:	string
	Default:	
compress	Compress calibration columns?	
	allowed:	bool
	Default:	false
addcorr	Add scratch columns?	
	allowed:	bool
	Default:	true
addmodel	Add MODEL_DATA column along with CORRECTED_DATA ?	
	allowed:	bool
	Default:	true

Returns
bool

Example

```
cb.open('ngc5921.ms');
```

calibrater.selectvis.html

calibrater.selectvis - Function

2.1.1 Set the data selection for subsequent processing

Description

This function provids for selection of the visibility data from the MS which will be treated by subsequent execution of the `solve` and `correct` functions. Note that data selection is not cumulative, i.e., any selection made in a previous call to `selectvis` will be overridden by the the current call.
Most of the `selectvis` parameters use the standardized MS Selection syntax. The parameters are described below. The selected data will satisfy the logical AND of all non-trivially specified parameters. Note that the old-fashioned strided channel selection parameters are deprecated (and will soon be removed); use spw instead. Running `selectvis` with no specified parameters restores selection of the entire MS.

time is used to specify time ranges in a stardard format

spw is used to specify spectral window and channel selection. Currently, only a single channel range can be specified per spw.

scan is used to specify scan numbers and ranges

observation is used to specify observation ID(s).

field is used to specify field names or indices

baseline is used to specify antenna and baseline combinations

uvrange is used to specify baseline length ranges

chanmode is deprecated (use spw)

nchan is deprecated (use spw)

start is deprecated (use spw)

step is deprecated (use spw)

mstart is deprecated (use spw)

mstep is deprecated (use spw)

msselect is used to specify a subselection of data according to Measurement Set columns in conditional combinations not possible with the standard parameters above. This parameter should be specified as a valid TaQL expression. If both msselect and the standard selection parameter are used together, they are combined with a logical AND, i.e., the data must jointly satisfy all `selectvis` parameters.

Arguments

Inputs			
time		Select on time	
		allowed:	any
		Default:	variant
spw		Select on spectral window	
		allowed:	any
		Default:	variant
scan		Select on scan	
		allowed:	any
		Default:	variant
field		Select on field	
		allowed:	any
		Default:	variant
intent		Select on intent or state	
		allowed:	any
		Default:	variant
observation		Select by observation ID(s)	
		allowed:	any
		Default:	variant
baseline		Select on antennas/baselines	
		allowed:	any
		Default:	variant
uvrange		Select by uvrange	
		allowed:	any
		Default:	variant
chanmode		Type of data selection: channel or velocity	
		allowed:	string
		Default:	channel
			velocity
			none
nchan		Number of channels to select (mode='channel')	
		allowed:	int
		Default:	1
start		Start channel (0-relative) (mode='channel')	
		allowed:	int
		Default:	0
step		Step in channel number (mode='channel')	
		allowed:	int
		Default:	1
mstart	km/s	Start velocity (e.g. '20Km/s')	
		allowed:	doublekm/s
		Default:	0.0
mstep	km/s	Step in velocity (e.g. '100m/s')	
		allowed:	doublekm/s
		Default:	0.0
msselect		TAQL selection string. Default (empty) is no specific selection.	
		allowed:	string

Returns
bool

Example

Open and select a field:

```
cb.open('ngc5921.ms');
cb.selectvis(field='N5921_2');  # by complete name
cb.selectvis(field='N5921*');   # with wildcard
cb.selectvis(field='2');        # by index
```

Select a field and a channel range:

```
cb.selectvis(spw='0:10~40',field='N5921*');
```

Select using all MS Selection parameters (these parameters are
over-specified somewhat, i.e., scan 6 contains only field N5921_2,
etc.):

```
cb.selectvis(time='>1995/04/13/10:40:00',   # times greater than this
             spw='0:20~40',                 # channels 20-40 in spw 0
             scan='6',                      # scan 6 only
             field='N59*',                  # fields matching N59*
             baseline='1 \& *',              # baselines to antenna 1
             uvrange='>0.0klambda')         # baselines greater than zero length
```

Reset selection to the entire dataset

```
cb.selectvis()
```

calibrater.setmodel.html

calibrater.setmodel - Function

2.1.1 Set the sky model used to compute the model visibilities

Description

Name of the model image to be used as a sky model for model visibility computations. For now, this is used only by EP-Jones solver.

Arguments

Inputs		
modelimage	Name of the model image.	
	allowed:	string
	Default:	

Returns

bool

Example

```
cb.setmodel("mymodel");
```

---

calibrater.setptmodel.html

calibrater.setptmodel - Function

2.1.1 Set the point source model Stokes parameters to be used to compute the model visibilities

Description

Set a global point source model Stokes parameters to use in solving operations.

Arguments

Inputs		
stokes	Vector of Stokes parameters.	
	allowed:	doubleArray
	Default:	0.0 0.0 0.0 0.0

Returns
bool

Example

```
cb.setmodel([1,1,0,0]);
```

---

calibrater.setapply.html

calibrater.setapply - Function

2.1.1 Arrange to apply calibration

Description

This function is used to specify the calibration components which should be applied during subsequent execution of the `solve` and `correct` functions. This function should be executed as many times as necessary to specify all desired calibration components.
Each calibration component represents a separate calibration matrix correction included in the measurement equation. The different types correspond to different instrumental and atmospheric effects. Calibration components are available as calibration tables generated by previous `solve` executions (types 'B','BPOLY','G','GSPLINE', 'D','DF','T','M','MF','X'), or are calculated analytically on the fly (types 'P', 'TOPAC', 'GAINCURVE'). Upon execution of `solve` or `correct`, the group of specified calibration components will be applied in the order prescribed by the Measurement Equation formalism.
The parameters are as follows:

type The calibration type being specified. This is only required for analytic types ('P','TOPAC','GAINCURVE'). When specifying an existing pre-solved calibration table, it is not necessary to explicitly specify the `type`; this will be discerned from the table. (Specifying the `type` as well as the `table` will force a check that the table contains solutions of the specified type.

For `type='GAINCURVE'`, an elevation-dependent correction will be applied using parameters read from the data repository. Currently, this is only supported for the VLA.

t This parameter will be used in a future release to control the range of applicability of the specified calibration. Currently, it is ignored.

table For pre-solved calibration, the file name of the table to apply.

field The fields to select from the specified table, using MS Selection syntax (as in selectvis).

interp The desired type of time-dependent interpolation. Use `interp='nearest'` to calibrate each datum with the calibration value nearest in time. Use `interp='linear'` to calibrate each datum with calibration phases and amplitudes linearly interpolated from neighboring

(in time) values. In the case of phase, this mode will assume that phase jumps greater than 180 degrees between neighboring points indicate a cycle slip, and the interpolated value will follow this change in cycle accordingly (i.e., the implied rate will always be less than 180 degrees per sample). Use `interp='aipslin'` to emulate the basic interpolation mode used in classic AIPS, i.e., linearly interpolated amplitudes, with phases derived from linear interpolation of the complex calibration values. While this method avoids having to track cycle slips (which is unstable for solutions with very low SNR), it will yield a phase interpolation which becomes increasingly non-linear as the spanned phase difference increases. The non-linearity mimics the behavior of `interp='nearest'` as the spanned phase difference approaches 180 degrees (the phase of the interpolated complex calibration value initially changes very slowly, then rapidly jumps to the second value at the midpoint of the interval). If the uncalibrated phase is changing this rapidly, a 'nearest' interpolation is not desirable. Usually, `interp='linear'` is the best choice. The `interp` parameter is applicable to any calibration type, as long as there are sufficient solutions available to perform the interpolation. Note that calibration solutions which have been determined for only one timestamp will default to 'nearest'. More interpolation options (e.g., 'cubic') will be added in the future.

select Used to specify general selection of a subset of calibration measurements from the table to be applied to the visibility data. Arbitrary cross-calibration is possible by combining this function with the `setdata` function. The string specified must be a valid TaQL expression.

spwmap This parameter is used to indicate how solutions derived from different spectral windows should be applied to other spectral windows. Nominally, data in each spectral window will be corrected by solutions derived from the same spectral window. This is the default behavior of `spwmap`, i.e., if `spwmap` is not specified, calibrater will insist that data be corrected by solutions from the same spw. Otherwise, `spwmap` takes a vector of integers indicating which spectral window *solutions* to apply to which spectral window *data*, such that `spwmap[j]=i` causes solutions derived from the i-th spectral window to be used to correct the j-th spectral window. For example, if (say) bandpass solutions are available for spws 0 & 2, and it is desired that these be applied to spws 1 & 3 (as well as 0 & 2), respectively, use `spwmap=[0,0,2,2]`. Even if some spws do not require an explicit `spwmap` setting, yet one or more does, it is safest to specify it explicitly for all, e.g., `spwmap=[0,1,3,3]` indicates that spw 2 will be corrected with solutions from spw 3, and the others will behave nominally. Note that if no solutions exist for any of the spws specified in `spwmap`, an error message will result.

calwt If set True, the data weights will be calibrated along with the data.

This is usually desirable.

opacity For `type='TOPAC'`, an elevation-dependent opacity correction will be applied according to the zenith opacity value supplied in the `opacity` parameter. Currently, only one zenith opacity value can be supplied, and it is used for all antennas.

Use the `state` function to review the list of calibration components that have been set for application.
Pending improvements:

- Enable variety of interpolation modes and timescales
- Allow for antenna- and time-dependent opacities

Arguments

Inputs		
type	Component type	
	allowed:	string
	Default:	B
		BPOLY
		G
		GSPLINE
		D
		P
		T
		TOPAC
		GAINCURVE
t	Interpolation interval (seconds)	
	allowed:	double
	Default:	0.0
table	Calibration table name	
	allowed:	string
	Default:	
field	Select on field	
	allowed:	any
	Default:	variant
interp	Interpolation type (in time)	
	allowed:	string
	Default:	aipslin
		nearest
		linear
select	TAQL selection string. Default is no selection.	
	allowed:	string
	Default:	
calwt	Calibrate weights?	
	allowed:	bool
	Default:	false
spwmap	Spectral windows to apply	
	allowed:	intArray
	Default:	-1
opacity	Array-wide zenith opacity per antenna (for type='TOPAC')	
	allowed:	doubleArray
	Default:	0.0

Returns
bool

Example

```
cb.open('ngc5921.ms')
cb.selectvis(field='N5921*')
cb.setapply (type='G', table='gcal', field='1445*')
cb.setapply (type='P')
cb.correct();
cb.close();
```

In this example, we apply parallactic angle corrections and a gain calibration derived from a field whose name matches '1445*' in a caltable called 'gcal' to data for a field matching 'N5921*'

---

calibrater.setcallib.html

calibrater.setcallib - Function

2.1.1 Arrange to apply calibration via a Cal Library

Description

TBD

Arguments

Inputs		
callib	A calibration library record	
	allowed:	record
	Default:	

Returns
bool

Example

```
TBD
```

---

calibrater.validatecallib.html

calibrater.validatecallib - Function

2.1.1 Validate a Cal Library record

Description

TBD

Arguments

Inputs		
callib	A calibration library record	
	allowed:	record
	Default:	

Returns
bool

Example

```
TBD
```

---

calibrater.setsolve.html

calibrater.setsolve - Function

2.1.1 Arrange to solve for calibration

Description

This function specifies the calibration component that will be solved for by the `solve` function. Currently, only one type can be solved for at one time. Each calibration component represents a separate calibration matrix correction included in the measurement equation. The different types correspond to different instrumental and atmospheric effects. Currently, the solvable calibration components are types 'G','T','B', 'D' and 'DF', which are antenna-based, and, 'M' and 'MF', which are baseline-based. Arrange to pre-apply any existing calibration components (of types other than the solved-for one) using the `setapply` function.
The parameters are:

type Specify the calibration type you want to solve for, from 'G','T','B','D','DF','M','MF'.

t Specify the solution interval. This can be specified as an integer (units of seconds assumed) or as a string containing a value and units (e.g., '30s', '45min', '2h') or 'inf' (infinite) or 'int' (per data integration). A solution interval of 0 (with or without units) is the same as 'int' (per integration), and negative solution intervals are treated as 'inf' (infinite).

table Specify the output calibration table name in which to store the calibration solve result. Existing tables will be deleted and replaced.

append Append the solutions to an existing table.

preavg Specify the amount of pre-average (in time) within the solution interval. By default, data are averaged up to the solution interval (or up to 5 minutes for 'D' solving).

phaseonly This parameter is deprecated, use apmode.

apmode Control generation of amplitude-only ('a'), phase-only ('p'), or amplitude-and-phase ('ap', the default) solutions.

refant Specify an antenna (using data selection syntax) for referencing the solutions.

solnorm Normalize the solutions by their mean post-solve. For 'B', and 'MF', this is a complex normalization per solution spectrum. For other types, this is a global (per-spw) normalization of the amplitudes only.

minsnr Specify the SNR below which solution are rejected.

combine Specify which data axes (spw, field, scan, or some combination) on which the data should be combined to generate a single solution. E.g., combine='spw' will force combination of many spws to form a single solution (per solution interval). Similarly, combine='scan' with a long solution interval will force the combination of scans to yield individual solutions (per field and spw). Ordinarily, solutions are always broken at scans boundaries. Separate multiple combine options with commas.

fillgaps For 'B' solutions, specify the largest solution channel gap (which arise due to flagged data) that will be filled post-solve via interpolation. Such solution gaps remain flagged by default.

Pending improvements:

- Change t to solint?
- Permit flexible specification of preavg (as for t)

Arguments

Inputs		
type	Component type	
	allowed:	string
	Default:	G
		T
		B
		D
		M
		MF
t	Solution interval (units optional)	
	allowed:	any
	Default:	variant
table	Output calibration table name	
	allowed:	string
	Default:	
append	Append to existing table?	
	allowed:	bool
	Default:	false
preavg	Pre-averaging interval (in sec)	
	allowed:	double
	Default:	-1.0
phaseonly	Solve only for phase?	
	allowed:	bool
	Default:	false
apmode	Solve for 'AP', 'A' (amp-only) or 'P' (phase-only)	
	allowed:	string
	Default:	AP
refant	Reference antenna. Default is none.	
	allowed:	any
	Default:	variant
minblperant	Minimum number of baselines per ant for solving	
	allowed:	int
	Default:	4
solnorm	Normalize solution after solve	
	allowed:	bool
	Default:	false
minsnr	SNR threshold for accepting solutions	
	allowed:	float
	Default:	0.0
combine	Data axes on which to combine solving (scan, spw, and/or field)	
	allowed:	string
	Default:	
fillgaps		
	allowed:	int
	Default:	0
cfcache	Name of the directory to be used for convolution function disk cache. This is used when type=EP.	
	allowed:	string
	Default:	
painc	Parallactic Angle increment used to trigger computation of a new convolution function. This is used when type=EP. Default value implies that only one convolution function will be computed for the entire range of	

Returns
bool

Example

```
cb.open('ngc5921.ms');
cb.setapply (type='P');
cb.setsolve (type='G',t='300s', refant=3, table='gcal');
cb.solve();
cb.close();
```

In this example, analytic (non-solvable) parallactic angle corrections are pre-applied before G solutions are obtained on a timescale of 300 seconds. The resulting solutions are phase-referenced to antenna 3, and stored in a calibration table called 'gcal'.

```
cb.reset();
cb.setapply (type='P',t=5.0);
cb.setapply (type='G',table='gcal');
cb.setsolve (type='D',t=86400.0, preavg=60.0, refant=3, table='dcal');
cb.solve();
cb.close();
```

In this example, the solve/apply state of the calibrater tool is reset and then the P and G corrections (from above) are applied before solving for D solutions on a diurnal timescale. Note that the data will be averaged only to 60 seconds before the solution. The resulting D solutions are stored in a table called 'dcal'.

---

calibrater.setsolvegainspline.html

calibrater.setsolvegainspline - Function

2.1.1 Specialization of setsolve for cubic spline G (time-dependent gain) solving

Description

This function is a specialization of the `setsolve` method which should be used when cubic spline G solutions are desired, e.g., when SNR on calibrators is very low. Currently, this solving mode treats dual polarization data on a per-polarization basis. The option to obtain a joint solution (a la 'T') will be provided in the future.
The visibility data are averaged in frequency (for multi-channel data) prior to the solution.
This method uses many of the basic parameters as the generic `setsolve`. Parameters unique to the spline solver are:

mode For phase solutions only, use `mode='PHAS'`. For amplitude solutions only, use `mode='AMP'`. If both are desired, use `mode='PHASAMP'`, and both will be solved for using the same spline timescale (this mode also assumes that all calibrators have the correct relative flux densities). If solving for phase and amplitude separately (usually in this order), it is usually desirable to apply the first one when solving for the second one. Spline solution so obtained will be stored in separate calibration tables. In the near future, the `mode` parameter will be consolidated with the generic `apmode` parameter.

splinetime The spline timescale (time between knots) is specified here. The default is 10800 seconds (3 hours). In future this parameter will be consolidated with the generic `t` parameter. The `preavg` parameter should be set to a value at least 4X shorter than the spline time (an error will occur if there is insufficient sampling within the `splinetime` timescale), and consistent with the expected coherence. Consistent with these constraints, use the largest possible value for `preavg` to optimize the SNR of the pre-solve phase-tracking algorithm.

npointaver and phasewrap These parameters tune the phase-unwrapping algorithm when `mode = 'PHAS'`. Cycle slips are detected (and removed before the spline solve) when the median phase a sequence of length `npointaver` (in integrations) differs by more than `phasewrap` degrees from the previous sequence.

Pending improvements:

- Consolidate more parameters with the generic `setsolve`
- Introduce the generic combine options
- Improve phase-tracking algorithm

Arguments

Inputs		
table	Output calibration table name	
	allowed:	string
	Default:	
append	Append to existing table?	
	allowed:	bool
	Default:	false
mode	Phase or Amplitude mode?	
	allowed:	string
	Default:	AMP
		PHASAMP
		PHAS
splinetime	Spline timescale (sec)	
	allowed:	double
	Default:	10800
preavg	Pre-averaging interval (in sec)	
	allowed:	double
	Default:	0.0
npointaver		
	allowed:	int
	Default:	10
phasewrap		
	allowed:	double
	Default:	250
refant	Reference antenna. Default is none.	
	allowed:	any
	Default:	variant

Returns
bool

Example

```
cb.open('ngc5921.ms')
cb.selectvis(field='1445*')
cb.setsolvegainspline (table='gcalph',mode='PHAS',splinetime=3600.0,preavg=60.0)cb.solve()

cb.setsolvegainspline (table='gcalamp',mode='AMP',splinetime=10800.0);
cb.solve();
cb.close();
```

In this example, a spline solution is first found for phase on a hourly timescale, then for

---

calibrater.setsolvebandpoly.html

calibrater.setsolvebandpoly - Function

2.1.1 Specialization of setsolve for polynomial B (bandpass) solving

Description

This function is a specialization of the `setsolve` method which should be used to arrange for bandpass solving when polynomial solutions for B are desired, e.g., when per-channel SNR on calibrators is too low to obtain a useful sampled bandpass.
Prior to the solution, the visibility data are averaged in time, and the solution is performed for both phase and amplitude.
This method uses most of the same parameters as the generic `setsolve`, with a few unique additions:

degamp and degphase The parameters permit specification of the polynomial order to use in amp and phase. Specifying 0 (zero) yields constant solutions.

visnorm This parameter is used to normalize the assembled spectral data, in a per baseline manner. If set True, this will have the effect of removing any non-frequency-dependent closure errors (e.g., as caused by source structure, or introduced by the instrument) from the data, and should be used with caution. The resulting solutions will be effectively normalized as well. When `visnorm=F` is used, closure errors in the data (as supplied to the solver) may be visible in the form of offsets between the data and solutions. For bandpass calibration, this is usually ok, as the *shape* of the bandpass is the most important aspect of the solution. In future this parameter will be generalized and made available for other solve types. (NB: Use of `solnorm=True` still provides for post-solve normalization of the solutions.)

maskcenter and maskedge These parameters control how many channels are ignored on-the-fly, at the center and edges of each input spectral window, respectively. To avoid edge channels, it is almost always better to flag these channels directly, or select against them in `setdata`. Aggressive use of maskedge (large values), will yield polynomial solutions which will tend to diverge at the edges (especially when the polynomial degree is also high), because maskedge does not change the frequency domain of the solutions. Such solutions should be used with caution in subsequent operations. (It is best to avoid use of maskedge.)

The BPOLY solution is performed for both phase and amplitude, and the result will be stored in the same table. The frequency domain of the solutions is limited to only the range of frequencies selected in `selectvis`. When correcting data with these solutions (for other solves or with `correct`), only data within this domain will be corrected. Data outside (e.g., edge channels avoided in `setdata` for the solve), will not be corrected. Therefore, the same (or narrower) channel selection is recommended for all operations using solutions produced by this function and `solve()`.
Note that the `combine` parmaeter can be used meaningfully with the BPOLY solver. When combine='spw', the data from multiple spws will be combined on a common frequency axis, and a single polynomial will be determined spanning them all. This is different than for ordinary sampled 'B' solutions, for which combine='spw' causes the bandpass to be combined on a common channel axis, effectively yielding a mean bandpass for the set of spws.

Arguments

Inputs		
table	Output calibration table name	
	allowed:	string
	Default:	
append	Append to existing table?	
	allowed:	bool
	Default:	false
t	Solution interval (units optional)	
	allowed:	any
	Default:	variant
combine	Data axes on which to combine solving (scan, spw, and/or field)	
	allowed:	string
	Default:	
degamp	Polynomial degree for amplitude solution	
	allowed:	int
	Default:	3
degphase	Polynomial degree for phase solution	
	allowed:	int
	Default:	3
visnorm	Normalize data prior to solution	
	allowed:	bool
	Default:	false
solnorm	Normalize result?	
	allowed:	bool
	Default:	true
maskcenter	Number of channels to avoid in center of each band	
	allowed:	int
	Default:	0
maskedge	Fraction of channels to avoid at each band edge (in %)	
	allowed:	double
	Default:	5.0
refant	Reference antenna	
	allowed:	any
	Default:	variant

Returns
bool

Example

```
cb.open('ngc5921.ms');
cb.selectvis(field='1331*')
cb.setsolvebandpoly(table='bpoly',degamp=5,degphase=7);
cb.solve();
cb.close();
```

In this example, amplitude (degree 5) and phase (degree 7) Chebychev polynomial bandpasses are determined using the default parameters.

---

calibrater.state.html

calibrater.state - Function

2.1.1 Request the apply/solve state of the calibrater tool

Description

Request the apply/solve state of the calibrater tool. A listing of all calibration components that have been set for application or solving is written to the logger.

Arguments

Returns
bool

Example

```
cb.open('ngc5921.ms');
cb.setapply ('P', 5.0);
cb.setsolve ('G', 300.0, F, 3, 'gcal_1', T);
cb.state();
```

---

calibrater.reset.html

calibrater.reset - Function

2.1.1 Reset the selected apply and/or solve components

Description

Resets the apply and/or solve components previously set by setapply and setsolve.

Arguments

Inputs		
apply	If true, unset all apply settings	
	allowed:	bool
	Default:	true
solve	If true, unset all solve settings	
	allowed:	bool
	Default:	true

Returns
bool

Example

```
cb.open('ngc5921.ms')
cb.setapply ('P', 5.0)
cb.setsolve ('G', 300.0, F, 3, 'gcal_1', T)
cb.state()
cb.reset(apply=T,solve=F);
cb.state()
cb.reset()
```

calibrater.initcalset.html

calibrater.initcalset - Function

2.1.1 Re-initialize the calibration scratch columns.

Description

This function re-initializes the calibration scratch columns: MODEL_DATA to unity (in total intensity, and unpolarized), and CORRECTED_DATA to (observed) DATA. Optionally if calset is set to 1 any model saved in the MS header to for calibration purposes is deleted

Arguments

Inputs		
calset	if it set to 1 the model saved in the header is removed	
	allowed:	int
	Default:	0

Returns

bool

Example

```
cb.open('ngc5921.ms');
cb.initcalset();
cb.solve();
```

calibrater.delmod.html

calibrater.delmod - Function

2.1.1 Delete model data representations in the MS.

Description

This method can be used to delete the model visibility data representations in the MS. The 'otf' representation is the new (as of v3.4) 'scratch-less' model data, stored as keywords in the MS header containing model data formation instructions. It is generated by the im tool (setjy, ft, and clean methods; usescratch=F in im.open), and if present, overrides the old-fashioned MODEL_DATA column (if present). If a user wishes to use the MODEL_DATA column _after_ having operated with the 'otf' representation, this method can be used to delete the 'otf' represenatation to make the MODEL_DATA column visible. (Create the MODEL_DATA column by using usescratch=T in the im tool, or by running the cb.open with addmodel=T.)
If otf=T, the user may selectively remove only a selection of fields model from the MS by specifying the field parameter. Similarly if the field parameter is specified, selected spws model for those fields may be deleted by specifying the spw.
For convenience, this method also provides a means for deleting the MODEL_DATA column by setting scr=T.

Arguments

Inputs		
otf	If T, delete the otf model data keywords	
	allowed:	bool
	Default:	false
field	Select on field	
	allowed:	any
	Default:	variant
spw	Select on spw only if field is defined	
	allowed:	any
	Default:	variant
scr	If T, delete the MODEL_DATA column	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
cb.open('ngc5921.ms');
cb.delmod(otf=T,scr=F);   # delete only the otf model for all fields
cb.solve();

cb.open('n4826.ms')
cb.delmod(otf=T, field='1')
#delete otf model of field 1 only, all other fields model are untouched
#if present
cb.open('n4826.ms')
cb.delmod(otf=T, field='1', spw='2')
#delete otf model of field 1 and spectralwindow 2  only.

####NOTE doing:
cb.delmod(otf=T, field='', spw='2')

#will delete all otf models and spw will be ignored
```

---

calibrater.solve.html

calibrater.solve - Function

2.1.1 Solve for the selected calibration components

Description

Execution of this function initiates a solve for the calibration component specified in a previous `setsolve` execution. Existing calibration components (as specified in one or more `setapply` executions) will be appropriately applied to the observed and model data according to their position in the Measurement Equation, and their commutation properties.

Arguments

Returns
bool

Example

```
cb.open('ngc5921.ms');
cb.setapply ('P', t=10)
cb.setsolve ('G', 300.0, F, 3, 'gcal_1', T);
cb.solve();
cb.close();
```

---

calibrater.correct.html

calibrater.correct - Function

2.1.1 Apply calibration information

Description

This function applies the calibration components specified via one or more invocations of the `setapply` function to the observed visibility data and writes the result to the CORRECTED_DATA column of the Measurement Set.

Arguments

Inputs		
applymode	Correction cal/flag mode: "='calflag','cal','flag','trial'	
	allowed:	string
	Default:	

Returns

bool

Example

```
cb.open('ngc5921.ms');
cb.selectvis(field='1445*')
cb.setapply ('G', 10.0, 'gcal_1')
cb.correct();
cb.close();
```

---

calibrater.corrupt.html

calibrater.corrupt - Function

2.1.1 Corrupt model with calibration tables

Description

This function applies the calibration components specified via one or more invocations of the `setapply` function to the model visibility data and (over-)writes the result to the MODEL_DATA column of the Measurement Set.

Arguments

Returns
bool

Example

```
cb.open('ngc5921.ms')
cb.selectvis(field='1445*')
cb.setapply ('G', 10.0, 'gcal_1')
cb.corrupt()
cb.close()
```

---

calibrater.initweights.html

calibrater.initweights - Function

2.1.1 Initialize MS weight info in various ways.

Description

This function initializes the MS weight info in various ways.
If wtmode='ones', SIGMA and WEIGHT will be initialized with 1.0, globally.
If wtmode='nyq' (the default), SIGMA and WEIGHT will be initialized according to bandwidth and integration time. This is the theoretically correct mode for raw normalized visibilities.
If wtmode='sigma', WEIGHT will be initialized according to the existing SIGMA column.
If mode='weight', WEIGHT_SPECTRUM will be initialized according to the existing WEIGHT column; dowtspec=T must be specified in this case.
For the above wtmodes, if dowtspec=T (or if the WEIGHT_SPECTRUM column already exists), the WEIGHT_SPECTRUM column will be initialized (uniformly in channel), in a manner consistent with the WEIGHT column. If the WEIGHT_SPECTRUM column does not exist, dowtsp=T will force its creation.
The follow modes should be used with extreme care: If wtmode='delwtsp', the WEIGHT_SPECTRUM column will be deleted (if it exists). If wtmode='delsigsp', the SIGMA_SPECTRUM column will be deleted (if it exists). Note that creation of SIGMA_SPECTRUM is not supported via this method.
Note that this method does not support any prior selection. Intialization of the weight information must currently be done globally or not at all. This is to maintain consistency.

Arguments

Inputs		
wtmode	Initialization mode	
	allowed:	string
	Default:	nyq
dowtsp	Initialize WEIGHT_SPECTRUM column	
	allowed:	bool
	Default:	false
tsystable	Tsys calibration table to apply on the fly	
	allowed:	string
	Default:	
gainfield	Select a subset of calibrators from Tsys caltable	
	allowed:	string
	Default:	
interp	Interp type in time[,freq]. default==linear,linear	
	allowed:	string
	Default:	
spwmap	Spectral windows combinations to form for gaintables(s)	
	allowed:	intArray
	Default:	

Returns
bool

Example

```
cb.open('ngc5921.ms')
cb.initweights()
cb.close()
```

---

calibrater.fluxscale.html

calibrater.fluxscale - Function

2.1.1 Bootstrap the flux density scale from standard calibrators

Description

This function is used to bootstrap the amplitude scale the calibration solutions according to specified reference calibrator(s) of known flux density. This is necessary when the flux densities of some of your calibrators were unknown (and thus were assumed to be 1 Jy) during G solving.
The bootstrapping is achieved by comparing the median gain norm of the calibration solutions derived for the calibrators specified in `reference` (one or more sources with known flux densities at the time of G solving) with that of the calibrators specified in `transfer`, and enforcing the assumption that the antenna gains are constant, on average. The gain solutions for the transfer sources are then re-scaled accordingly. The `reference` and `transfer` parameters may be specified using the general field selection syntax (as in `field` in `selectvis`).
If no `transfer` fields are specified, then the solutions for all non-reference fields in `tablein` will be re-scaled.
If no `tableout` is specified the input table will be overwritten with the scaled solutions. Note that the resulting table will only contain solutions for those fields implicit in the `reference` and `transfer` specifications. Use `append=T` to append the scaled solutions to an existing table.
Use the `refspwmap` parameter to indicate how data for different spectral windows should be matched in calculating the flux density scale factor for `transfer` fields. The default behavior for `refspwmap` is to insist on precisely matching spectral windows for `reference` and `transfer` fields. When specified, the `refspwmap` parameter takes a vector of integers indicating which spectral window solutions to use as the reference for others, such that `refspwmap[j]=i` causes solutions (from reference fields) observed in the i-th spectral window to be used to reference solutions (from transfer fields) observed in the j-th spectral window. For example, for the case of a total of 4 spectral windows: if the `reference` fields were observed only in spw=2 & 4, and the `transfer` fields were observed variously in all 4 spws, specify `refspwmap=[2,2,4,4]`. This will ensure that `transfer` fields observed in spws 1,2,3,4 will be referenced to `reference` field data from spws 2,2,4,4, respectively. Note that if the `transfer` fields were observed only in spws 1 & 3, the same specification would work, but `refspwmap=[2,2,4]` would suffice. In this case, nothing need be specified for the 4th spw (there are no transfer fields

there), and specifying 2 for the 2nd spw is actually inconsequential (though required so that the specification of 4 for spw 3 is properly interpretted).
The gain values used in the flux scaling determination skewed by outliers. The parameters, `gainthreshold` and `antenna` can be used to limit the input gain solutions to be included in the flux scale determination. Use the `gainthreshold` is a threshold in % from the median values of the gain solutions to be used. Use the `antenna` to select or de-selesect (using the MSSelection syntax) antenna(s). Futher refinements on the selection based on timerange and scan are possible.
The derived flux densities for the transfer fields will be reported in the logger, and returned to the Python dictionary specified in `fluxd`. This will be an 2D array of shape [number-of-spectral-windows X number-of-fields]. When mulitple spectral windows are involved the spectral index will also be reported by fitting the determined flux densities across the freuquencies. The order of a polynomcial to be fitted can be specified with `fitorder`.
Note that elevation-dependent gain effects may render the basic assumption used here invalid, and so should be corrected for prior to solving for G, using types 'TOPAC' or 'GAINCURVE' in `setapply`.
Note that the visibility data itself is not used directly by this function.
Pending improvements:

- Allow antenna and uv-distance selection to improve results for resolved calibrators
- Set the visibility model according to the flux density results
- An option to use the data to derive the relative flux densities

Arguments

Inputs	
tablein	Input calibration table name allowed: string Default:
reference	Reference calibrator field names (comma-separated) allowed: any Default: variant
tableout	Output calibration table name. Default is input calibration table name. allowed: string Default:
transfer	Transfer source field names (comma-separated). Default is all other fields. allowed: any Default: variant
listfile	Name of listfile that contains the fit information. Default is ” (no file). allowed: string Default:
append	Append to existing table? allowed: bool Default: false
refspwmap	List of alternate spw for referencing allowed: intArray Default: -1
gainthreshold	Threshold of gain amplitudes with respect to the median value to be used in flux scale calculation. Default: -1.0 (no threshold) allowed: float Default: -1.0
antenna	antenna selection/de-selection in flux scale calculation. Default: ””(include all antennas) allowed: string Default:
timerange	timerage sub-selection with antenna selection in flux scale calculation. Default: ””(include all) allowed: string Default:
scan	scan sub-selection with antenna selection in flux scale calculation. Default: ””(include all) allowed: string Default:
incremental	create a incremental caltable allowed: bool Default: false
fitorder	order for spectral fitting for multiple spws allowed: int Default: 1
display	display statistics of the flux ratios allowed: bool Default: false

Returns
record

Example

```
cb.open('ngc5921.ms')
cb.selectvis(field='1331*,1445*')
cb.setsolve(type='G',table='gcal',t='inf')
cb.solve()
cb.fluxscale (tablein='gcal', tableout='flxcal',
              reference='1331*', transfer='1445*');
cb.close();
```

```
This example generates a calibration table containing {\tt G}
solutuions ('gcal') and then writes a re-scaled version, using
1335+305 as the reference calibrator, to derive properly scaled
amplitude calibration for the transfer source, 1445+099.  We have
assumed that 1331+305 has already had its MODEL\_DATA set to
the correct flux density.
```

---

calibrater.accumulate.html

calibrater.accumulate - Function

2.1.1 Accumulate incremental calibration solutions into a cumulative calibration table

Description

This function enables cumulative calibration using `calibrater`. It is the analog of the task "CLCAL" in classic AIPS.
The `accumulate` function is useful when:

- a calibration solution of a particular type already exists,
- an incremental calibration solution *of the same type* is desired (an incremental solution in this context means derived independently from, or determined with respect to, the first)
- the first calibration cannot be implicitly recovered in the course of obtaining the incremental solution

For example, a phase-only "G" self-calibration on a target source may be desired to tweak the full amplitude and phase "G" calibration already obtained from a calibrator. The initial calibration (from the calibrator) contains amplitude information, and so must be carried forward, yet the phase-only solution itself cannot (by definition) recover this information, as a full amplitude and phase self-calibration would. In this case, the initial solution must be applied while solving for the phase-only solution, then the two solutions combined to form a *cumulative* calibration embodying the net effect of both. In terms of the Measaurement Equation, the net calibration is the *product* of the initial and incremental solutions.
The analog of `accumulate` in classic AIPS is the use of CLCAL to combine a series of (incremental) SN calibration tables to form successive (cumulative) CL calibration tables.
Cumulative calibration tables also provide a means of generating carefully interpolated calibration, on variable user-defined timescales, that can be examined prior to application to the data with `setapply` and `correct`. The solutions for different fields and/or spectral windows can be interpolated in different ways, with all solutions stored in the same table.
The only difference between incremental and cumulative calibration tables is that incremental tables are generated directly from the data via `solve` or (in the near future) from other ancilliary data (e.g. weather information), and cumulative tables are generated from other cumulative and incremental tables

via `accumulate`. In all other respects (internal format, application to data via `setapply` and `correct`, plotting with `plotcal`, etc.), they are the same, and therefore interchangable. Thus, `accumulate` and cumulative calibration tables need only be used when circumstances require it.

The `accumulate` function represents a generalization on the classic AIPS CLCAL model of cumulative calibration in that its application is not limited to accumulation of "G" solutions (SN/CL tables classic AIPS are the analog of "G" (and, implicitly, "T") in `aips++`). In principle, any basic calibration type can be accumulated (onto itself), as long as the result of the accumulation (matrix product) is of the same type. This is true of all the basic types, except "D". Accumulation is currently supported for "B", "G", and "T", and, in future, "F" (ionospheric Faraday rotation), "J" (generic full-polarization calibration), fringe-fitting, and perhaps others. Accumulation of certain specialized types (e.g., "GSPLINE", "TOPAC", etc.) onto the basic types will be supported in the near future. The treatment of various calibration from ancilliary data (e.g., system temperatures, weather data, WVR, etc.), as they become available, will also make use of `accumulate` to achieve the net calibration.

Note that accumulation only makes sense if treatment of a uniquely incremental solution is required (as described above), or if a careful interpolation or sampling of a solution is desired. In all other cases, re-solving for the type in question will suffice to form the net calibration of that type. For example, the product of an existing "G" solution and an amplitude and phase "G" self-cal (solved with the existing solution applied), is equivalent to full amplitude and phase "G" selfcal (with no prior solution applied), as long as the timescale of this solution is at least as short as that of the existing solution.

Use of `accumulate` is straightforward:

The `tablein` parameter is used to specify the existing cumulative calibration table to which an incremental table is to be applied. Initially, no such table exists, and `accumulate` will generate one from scratch (on-the-fly), using the timescale (in seconds) specified by the parameter `t`. These nominal solutions will be unit-amplitude, zero-phase (i.e., unit matrix) calibration, ready to be adjusted by accumulation. When `t` is negative (the default), the table name specified in `tablein` must exist and will be used.

The `incrtable` parameter is used to specify the incremental table that should be applied to `tablein`. The calibration type of `incrtable` sets the type assumed in the operation, so `tablein` must be of the same type. If it is not, `accumulate` will exit with an error message. (Certain combinations of types and subtypes will be supported by accumulate in the future.)

The `tableout` parameter is used to specify the name of the output table to write. If un-specified (or ""), then `tablein` will be overwritten. Use this feature with care, since an error here will require building up the cumulative table from the most recent distinct version (if any).

The `field` parameter specifies those field names (standard selection syntax) in `tablein` to which the incremental solution should be applied. The solutions

for other fields will be passed to `tableout` unaltered. If the cumulative table was created from scratch in this run of `accumulate`, then these solutions will be unit-amplitude, zero-phase, as described above.
The `calfield` parameter is used to specify the fields (standard selection syntax) to select from `incrtable` to use when applying to `tablein`. Together, use of `field` and `calfield` permit completely flexible combinations of calibration accumulation with respect to fields. Multiple runs of `accumulate` can be used to generate a single table with many combinations. In future, a "self" mode will be enabled that will simplify the accumulation of field-specific solutions.
The `interp` parameter is used to specify the interpolation type to use on the incremental solutions, as in `setapply`. The currently available interpolation types are "nearest", "linear", and "aipslin". See the `setapply` URM documentation for more details.
The `spwmap` parameter enables accumulating solutions from differing spectral windows. See `setapply` for details on how spwmap works.
Pending improvements:

- Implement a "self" mode (independent of interpolation type), to simplify or eliminate use of the `field` and `calfield` parameters in some contexts (e.g., self-cal)
- More interpolation modes, e.g., "cubic", and interpolation timescale (timerange to permit interpolation)
- Handle propogation (or not) of bad/flagged solutions
- Support of specialized types (e.g., TOPAC) onto the basic types
- Smoothing (probably a separate function)

Arguments

Inputs		
tablein	Input cumulative calibration table name	
	allowed:	string
	Default:	
incrtable	Input incremental calibration table name	
	allowed:	string
	Default:	
tableout	Output cumulative calibration table name. Default is input table name.	
	allowed:	string
	Default:	
field	List of fields (names) to update in input cumulative table. Default is all.	
	allowed:	any
	Default:	variant
calfield	List of fields (names) in incremental table to use. Default is use all.	
	allowed:	any
	Default:	variant
interp	Interpolation mode to use on incremental solutions	
	allowed:	string
	Default:	linear
t	Cumulative table timescale when creating from scratch	
	allowed:	double
	Default:	-1.0
spwmap	Spectral windows to apply	
	allowed:	intArray
	Default:	-1

Returns
bool

Example

```
cb.open('ap366.sim');

# obtain G solutions from calibrator
cb.selectvis(msselect='FIELD_ID IN [9,11]');
```

```
cb.setsolve(type='G',table='cal.G0',t=300);
cb.solve()

# obtain proper flux density scale
cb.fluxscale (tablein='cal.G0', tableout='cal.G1',
             reference='1328+307', transfer="0917+624");

# generate cumulative table for target source on 20s timescale
cb.accumulate(tablein='',incrtable='cal.G1',tableout='cal.cG0',
             field='0957+561',calfield='0917+624',
             interp='linear',t=20);

# apply this calibration to target
cb.selectvis(msselect='FIELD_ID==10');
cb.setapply(type='G',table='cal.cG0',interp='linear')
cb.correct();

#    (image target with imager tool)

# phase-selfcal target on 60s timescale
cb.selectvis(msselect='FIELD_ID==10');
cb.setapply(type='G',table='cal.cG0',interp='linear')
cb.setsolve(type='G',table='cal.G2',t=60,phaseonly=T);
cb.solve();

# accumulate new solution onto existing one
cb.accumulate(tablein='cal.cG0',incrtable='cal.G2',tableout='cal.cG1',
             field='0957+561',calfield='0957+561',
             interp='linear');

# apply new cumulative solution to data
cb.setapply(type='G',table='cal.cG1',interp='linear')
cb.correct();

#   (another round of imaging, etc.)

cb.close();
```

calibrater.activityrec.html

calibrater.activityrec - Function

2.1.1 Returns a record containing properties of recent activity

Description

This funtion enables returning generic information about recent activity.
Pending improvements:

- ??

Arguments

Returns
record

Example

```
TBD
```

---

calibrater.specifycal.html

calibrater.specifycal - Function

2.1.1 Externally specify calibration of various types

Description

This function enables specifying calibration parameters externally.

Arguments

Inputs		
caltable	The calibration table name	
	allowed:	string
	Default:	
time	Calibration timestamp	
	allowed:	string
	Default:	
spw	Calibration spw(s)	
	allowed:	string
	Default:	
antenna	Calibration antenna(s)	
	allowed:	string
	Default:	
pol	Calibration polarization	
	allowed:	string
	Default:	
caltype	Calibration timestamp	
	allowed:	string
	Default:	
parameter	Calibration parameters	
	allowed:	doubleArray
	Default:	1.0
infile	Ancillary input file	
	allowed:	string
	Default:	

Returns
bool

Example

```
cb.open('ap366.sim');

(TBD)

cb.close();
```

---

calibrater.smooth.html

calibrater.smooth - Function

2.1.1 Produce a smoothed calibration table

Description

This function provides for time-dependent smoothing of sampled calibration solutions. Currently supported types are 'G', 'B', and 'T'. (Smoothing on the frequency axis for 'B' will be supported in the near future.)
Two (sliding) smoothing types are currenlty supported: 'median' or 'mean', one of these options should be specified in `smoothtype`. The full width (in seconds) of the smoothing filter should be specified in `smoothtime`. Amplitude and (ambiguity-corrected) phase are smoothed separately.
Use `field` to limit the smoothing operation to a subset of the fields (standard selection syntax) found in the calibration table (other fields will pass to the output table unsmoothed). If `field` is left blank, all fields in the table will be smoothed.
The smoothing is always done independently for each field, but scan boundaries are not observed. Thus, if the `smoothtime` is large enough, smoothing may occur over many boundaries.
Flagged solutions in the input table will not participate in the smoothing calculation, but will be replaced with smoothed values if the smoothing window covers one or more unflagged solutions when centered on the flagged point.
Pending improvements:

- Add other smoothtypes?
- Add spw and other selection on input table
- Add A/P toggle

Arguments

Inputs		
tablein	Input calibration table	
	allowed:	string
	Default:	
tableout	Output calibration table	
	allowed:	string
	Default:	
field	Limit smoothing to these fields (default is all fields)	
	allowed:	any
	Default:	variant
smoothtype	The smoothing type: 'mean' or 'median'	
	allowed:	string
	Default:	mean
		median
smoothtime	Smoothing filter time constant (sec)	
	allowed:	double
	Default:	60.0

Returns
bool

Example

```
cb.open('ngc5921.ms');
cb.smooth(tablein='in.gcal',tableout='out.gcal',
         smoothtype='median',smoothtime=60);
cb.close();
```

```
In this example, 'G' solutions for all fields in the table 'in.gcal' are
smoothed using a median filter with a full-width of 60 seconds,
and the result written to 'out.gcal'.
```

calibrater.listcal.html

calibrater.listcal - Function

2.1.1 List the contents of a calibration table

Description

calibrater.listcal() lists antenna gain solutions in tabular form. The table is organized as follows. Solutions are output by

1. Spectral window,
2. Antenna,
3. Time,
4. Channel,
5. and Polarization.

The inner-most loop is over polarization. A "Spw Header" row is printed each time the spectral window changes. In addition to listing the spectral window ID (SpwID), the Spw Header also lists the date of observation (Date), the calibration table name (CalTable), and the measurement set name (MS name). A lower-level "antenna header" is printed each time the antenna names change or every 'pagerows' of output, whichever comes first. The antenna header column are described here:

Column Name	Description
Ant	Antenna name
Time	Visibility timestamp corresponding to gain solution
Field	Field name
Chn	Channel number
Amp	Complex solution amplitude
Phs	Complex solution phase
F	Flag

Elements of the "F" column contain an 'F' when the datum is flagged, and ' ' (whitespace) when the datum is not flagged.
Presently, the polarization mode names (for example: R, L) are not given, but the ordering of the polrization modes (left-to-right) is equivalent to the order output by task listobs (see "Feeds" in listobs output).

Arguments

Inputs		
caltable	Calibration table to list	
	allowed:	string
	Default:	
field	Field names or indices to list: "==>all	
	allowed:	any
	Default:	variant
antenna	Antenna/Baseline to list: "==>all	
	allowed:	any
	Default:	variant
spw	Spectral windows and channels: "==>all, spw='10:8~20'	
	allowed:	any
	Default:	variant
listfile	Send output to file: "==>send to terminal)	
	allowed:	string
	Default:	
pagerows	Rows per page	
	allowed:	int
	Default:	50

Returns
bool

Example

Input:

The following example imports a UVFITS file, performs a bandpass calibration, and displays a subset of the resulting calibration table.

```
pathname=os.environ.get('CASAPATH').split()[0] # Get path to CASA home dir
fitsdata=pathname+'/data/demo/NGC5921.fits' # Select uv-data (FITS) file
msdata='NGC5921.ms' # MS name; write to current directory
importuvfits(fitsfile=fitsdata, vis=msdata) # import FITS data to MS
setjy(vis=msdata) # Create model data for flux calibrator
```

```
caldata=msdata+'.bcal' # Calibration table name
bandpass(vis=msdata, caltable=caldata) # Bandpass calibration
cb.open(msdata) # Open MS in cb
cb.listcal(caltable=caldata, field='N5921_2, 0, 1', antenna='1~5;10~13;20~22', spw='0:4~6',

Output:

SpwID = 0, Date = 1995/04/13,  CalTable = NGC5921.ms.bcal (B Jones), MS name = /users/jcross
```

			Ant = 1				Ant = 2			
Time	Field	Chn	Amp	Phs	F	Amp	Phs	F	Amp	Phs	F
09:21:46.0	1331+30500002_0	4	0.294	5.3		0.264	3.5		0.296	105.9		0.287	-111.9	
09:21:46.0	1331+30500002_0	5	0.303	5.3		0.279	0.6		0.305	107.0		0.298	-111.6	
09:21:46.0	1331+30500002_0	6	0.307	5.6		0.287	-1.6		0.309	107.5		0.303	-111.5	
10:05:27.9	1445+09900002_0	4	0.467	7.6		0.419	2.7		0.473	107.7		0.455	-112.3	
10:05:27.9	1445+09900002_0	5	0.472	7.3		0.440	0.0		0.486	109.1		0.471	-111.8	
10:05:27.9	1445+09900002_0	6	0.486	8.4		0.453	-2.4		0.482	110.0		0.478	-111.4	
10:09:05.3	N5921_2	4	0.082	50.0		0.074	34.7		0.097	-74.5		0.083	54.4	
10:09:05.3	N5921_2	5	0.074	62.7		0.084	24.3		0.114	-73.1		0.093	47.3	
10:09:05.3	N5921_2	6	0.079	44.4		0.081	21.7		0.092	-66.3		0.101	48.0	

			Ant = 5				Ant = 10			
Time	Field	Chn	Amp	Phs	F	Amp	Phs	F	Amp	Phs	F
09:21:46.0	1331+30500002_0	4	0.261	-26.0		0.285	-107.1		0.279	-149.5		0.263	2.9	
09:21:46.0	1331+30500002_0	5	0.269	-26.1		0.295	-107.2		0.288	-148.8		0.274	2.3	
09:21:46.0	1331+30500002_0	6	0.272	-26.1		0.300	-107.0		0.293	-148.7		0.281	2.0	
10:05:27.9	1445+09900002_0	4	0.416	-24.0		0.450	-106.4		0.437	-147.3		0.414	3.2	
10:05:27.9	1445+09900002_0	5	0.421	-22.6		0.478	-106.1		0.453	-147.4		0.433	2.0	
10:05:27.9	1445+09900002_0	6	0.436	-22.7		0.478	-106.7		0.459	-146.6		0.443	2.4	
10:09:05.3	N5921_2	4	0.074	95.0		0.085	-4.2		0.083	109.6		0.084	-116.6	
10:09:05.3	N5921_2	5	0.071	96.8		0.084	-13.7		0.086	104.3		0.100	-116.4	
10:09:05.3	N5921_2	6	0.082	84.9		0.078	-5.3		0.101	109.1		0.102	-109.9	

			Ant = 13				Ant = 20			
Time	Field	Chn	Amp	Phs	F	Amp	Phs	F	Amp	Phs	F
09:21:46.0	1331+30500002_0	4	0.285	-169.5		0.277	-90.4		0.254	-36.0		0.290	70.9	
09:21:46.0	1331+30500002_0	5	0.296	-168.9		0.287	-90.5		0.269	-38.6		0.300	71.0	
09:21:46.0	1331+30500002_0	6	0.301	-168.7		0.291	-90.3		0.278	-39.9		0.306	70.9	
10:05:27.9	1445+09900002_0	4	0.448	-167.2		0.434	-90.7		0.401	-34.4		0.456	70.8	
10:05:27.9	1445+09900002_0	5	0.466	-166.5		0.457	-91.1		0.423	-36.6		0.470	70.8	
10:05:27.9	1445+09900002_0	6	0.473	-166.9		0.464	-91.1		0.436	-37.8		0.485	70.2	
10:09:05.3	N5921_2	4	0.097	83.0		0.087	143.5		0.080	43.1		0.094	144.2	
10:09:05.3	N5921_2	5	0.100	87.4		0.094	137.0		0.061	54.7		0.098	153.3	
10:09:05.3	N5921_2	6	0.099	93.3		0.122	140.5		0.077	51.3		0.090	151.7	

```
Listed 108 antenna solutions.
```

---

calibrater.posangcal.html

calibrater.posangcal - Function

2.1.1 Apply position angle calibration to an existing cal table

Description

This function is used to apply position angle calibration for observations made using circularly polarized feeds. According to the Measurement Equation formalism, this correction should be applied to a `D` (instrumental polarization) calibration table.
If no `D` calibration is performed (and thus no such table is available), the correction can be applied to a `G` table, but it should NEVER be applied to both, and always applied to a `D` table if one is available. An input table must be specified. If no output table is specified, then the input table will be modified in place.
Specify, as a vector of values, a position angle adjustment (in degrees) for each spectral window. If only one value is specified, it will be duplicated to all spectral windows; otherwise, the number of values specified must match the number of spectral windows. The sign convention for the position angle adjustment is such that the specified value is the that which, when added to the position angle implied by the data, will yield the correct position angle. For example, if `G-`, `D-`, and `P-`calibrated data for 3c286 suggests a position angle of 45 degrees, the posangcor value should be -12 degrees as this will yield the correct position angle of 33 degrees when added. In general, posangcor equals correct position angle minus observed position angle.
A future version of this function will have an option to recognize standard position angle calibrators and determine the correction automatically.
(NB: It may be desirable to use solutions for 'X' to handle position angle calibration, rather than this method.)

Arguments

Inputs		
posangcor	Position angle corrections (degrees)	
	allowed:	doubleArray
	Default:	
tablein	Input calibration table name	
	allowed:	string
	Default:	
tableout	Output calibration table name. Default is input table name.	
	allowed:	string
	Default:	

Returns
bool

Example

```
cb.open('polcal.ms');
cb.posangcal(tablein='3C286.dcal', tableout='3C286.dpacal',
             posangcor=[-12.0, 54.0]);
cb.close();
```

This example takes an existing calibration table containing {\tt D} Jones matrices, and applies a position angle calibration of 45 and 54 degrees to spectral windows 1 \& 2, respectively, writing the result to a new table. The observed position angles for 3C286 must have been 45 and -21 degrees; the corrections specified yield the correct value of 33 degrees when added to the observed values.

---

calibrater.linpolcor.html

calibrater.linpolcor - Function

2.1.1 Correct the gain table for linear polarization of the calibrator

Description

THIS METHOD IS CURRENTLY DISABLED.
This function can be used to correct the gains derived from secondary calibrators with unknown or variable polarization. It should only be used for arrays with linear (X/Y) feeds and an Alt-Az mount for which the observed polarization varies with feed position angle on the sky.
The function fits the gains with a sine and cosine term in feed position angle and extracts the Q and U components of the secondary calibrator. This is only possible if there is sufficient range in the position angle (i.e., minimum of about 6 scans spanning at least 90 degrees in position angle). Check the error of the fit to judge if the fit was succesfull, it should generally be smaller than 0.5%.
Use the `fields` argument to select calibrator fields to be fitted. The function takes a calibration table as input, and can write the adjusted gain solutions to the same table on output, or create a new table containing these results. The function also prints the derived polarization for each field for each spectral window.

Arguments

Inputs		
tablein	Input calibration table name	
	allowed:	string
	Default:	
tableout	Output calibration table name	
	allowed:	string
	Default:	
fields	Calibrator field names	
	allowed:	stringArray
	Default:	

Returns
bool

Example

```
cb.open('atca.ms');
cb.linpolcor(tablein='atca.gcal', tableout='atca.gcal2',
             fields='2254-367');
cb.close();
```

This example takes an existing calibration table containing {\tt G} Jones matrices, and writes a corrected output table, correcting only gains derived from 2254-367 for linear polarization.

---

calibrater.plotcal.html

calibrater.plotcal - Function

2.1.1 Plot a calibration table

Description

This function plots a calibration table either to a plotter or to a file.
The argument `plottype` can take the following values for all types of solutions:

AMP Gain Amplitude vs. Time

1/AMP Inverse Gain Amplitude vs. Time (useful for comparing with classic AIPS)

PHASE Gain Phase vs. Time

RI Gain Real vs. Imaginary

RLPHASE Right/Left Gain phase difference (if polarizations are R,L)

XYPHASE X/Y Gain phase difference (if polarizations are X,Y)

The argument `plottype` can take the following values for D tables

DAMP Cross-polarized Gain Amplitude vs. Time

DPHASE Cross-polarized Gain Phase vs. Time

DRI Cross-polarized Gain Real vs. Imaginary

The quality of the solutions can be examined with the following `plottype` choices:

FIT Fit per spectral window

FITWGT Fit weight per spectral window

TOTALFIT Total fit

By default, all antennas (as specified in the antennas argument) will appear on the same plot. Separate plots (all with the same scale) for each antenna can be activated by setting multiplot=T. The multiplot argument only separates plots by antenna (not, e.g., by the field_id(s) specified in the fields argument). If multiplot=T, the nx and ny arguments can be used to specify the number of plots per page.

At the moment, only one polarization can be plotted per execution. This restriction will be relaxed in the near future.
For B solutions, the plotting will loop over timestamps (if more than one).
A hardcopy plot can be created by specifying the `psfile` argument (which is especially useful for batch processing when a display screen is not available). This will cause the plot to be written to a PostScript file which can be subsequently sent to a printer.

Arguments

Inputs		
antennas	Antennas to plot. Default is none.	
	allowed:	intArray
	Default:	
fields	Fields to plot. Default is none.	
	allowed:	intArray
	Default:	
spwids	Spectral windows id.'s to plot. Default is none.	
	allowed:	intArray
	Default:	
plottype	Plot type	
	allowed:	string
	Default:	1/AMP
		PHASE
		RLPHASE
		XYPHASE
		RI
		DAMP
		DPHASE
		DRI
		FIT
		FITWGT
		TOTALFIT
		AMP
tablename	Calibration table name	
	allowed:	string
	Default:	
polarization	Polarization to plot	
	allowed:	int
	Default:	1
multiplot	Turn on separate antenna plots	
	allowed:	bool
	Default:	false
nx	If multiplot=T, number of plots on horizontal axis	
	allowed:	int
	Default:	1
ny	If multiplot=T, number of plots on vertical axis	
	allowed:	int
	Default:	1
psfile	Name of output PostScript file to write plot to. Default is to send plot to the screen.	
	allowed:	string
	Default:	

Returns

bool

Example

```
cb.open('ngc5921.ms');
cb.plotcal(plottype='PHASE', tablename="gcal", antennas=[1,3], polarization=2);
cb.close();
```

---

calibrater.modelfit.html

calibrater.modelfit - Function

2.1.1 Model fitting

Description

This method fits single-component models (points, elliptical Gaussians or elliptical Disks_ to the CORRECTED_DATA of the selected field. A first guess for the component parameters may be specified in the `par` parameter.

Arguments

Inputs	
vary	If specified where T, let this parameter (in par) vary in fit allowed: boolArray Default:
niter	Number of non-linear fitting iterations allowed: int Default: 0
compshape	Component shape, P=point G=gaussian allowed: string Default: P
par	Initial guess for fit parameters (default is for "P)"I flux, rel RA, rel Dec, 1,0, 0.0, 0.0 are defaults allowed: doubleArray Default: 1.0 0.0 0.0
file	If specified, output componentslist file name, if empty don't write componentslist file allowed: string Default:

Returns
doubleArray

Example

```
cb.open('ngc5921.ms');
cb.selectvis(field='1331*')
cb.modelfit(compshape='P',par=[15.0,0.0,0.0])
cb.close();
```

This example fits a point source mode using 15.0 Jy at the origin (phase center) as a first guess.

```
cb.open('ngc5921.ms');
cb.selectvis(field='1331*')
cb.modelfit(compshape='G',par=[15.0,0.0,0.0,2.0,1.0,0.0])
cb.close();
```

This example fits a Guassian model with a starting guess of 15 Jy at the phase center (0,0), with 2.0 arcsec major axis, 1.0 axial ratio, at position angle 0.0 deg.

---

calibrater.updatecaltable.html

calibrater.updatecaltable - Function

2.1.1 Caltable modernizer.

Description

This method can be used to update a caltable (from v3.4 or later) to the current version of CASA.
The following updates are currently supported.
o At CASA v4.1.0, the OBSERVATION subtable and OBSERVATION_ID column were added to caltables. This method adds trivial versions of these elements to pre-v4.1 caltables.

Arguments

Inputs		
caltable	Name of the caltable.	
	allowed:	string
	Default:	

Returns
bool

Example

```
cb.updatecaltable("mycaltable");
```

---

calibrater.close.html

calibrater.close - Function

2.1.1 Close the calibrater tool

Description

Close the `calibrater` tool, which is hardly ever necessary.

Arguments

Returns
bool

Example

```
cb.open('ngc5921.ms');
cb.close();
```

---

calibrater.done.html

calibrater.done - Function

2.1.1 Destroy the calibrater tool

Description

This function is redundant with the `close` method.

Arguments

Returns
bool

Example

```
cb.open('ngc5921.ms');
cb.done();
```

---

calanalysis-Tool.html

2.1.2 calanalysis - Tool

Get and fit data from a calibration table (CASA 3.4 and later).
Requires:

Synopsis

Description

Overall Description

The calibration analysis (ca) tool is a standardized interface to the new format (CASA 3.4 and later) calibration tables. It is designed to handle all types of tables, e.g., gain, bandpass, Tsys, etc. The ca tool takes advantages of newly implemented features in the CASA C++ code tree, e.g., iteration and parameter selection, which means that calibration tables can be accessed in a manner very similar to measurement sets.
The ca tool was originally designed to facilitate getting and/or processing data from an entire calibration table in an organized fashion so that the information could be written to other files. Additional features, e.g. introspective member functions, were added so that scripters and general users can easily employ the ca tool without using iteration (get one piece of data at a time).
Like the imaging tool (im), image analysis tool (ia), calibration tool (cb), etc., a native calibration analysis tool (ca) is created when CASA starts up. Other instances of the calibration analysis tool can be created, if required, using the command:

from casac import casac caLoc = casac.calanalysis()

Open/Close Member Functions

The purpose of the open() and close() member functions is obvious, i.e., they open and close the new format calibration table. Each ca tool instance can have only one open table. If no table has been opened, the other member functions don't return anything.
The member function definitions are:
ca.open('¡caltable name¿') - This member function opens the calibration table. If successful True is returned, otherwise False is returned.
ca.close() - This member function closes the calibration table. If a table was open True is returned, otherwise False is returned.

Introspective Member Functions

The introspective member functions provide information about the shape and contents of the file. For example, the numchannel() member function returns the number of channels corresponding to each spectral window. Also, the field() member function returns the field names or numbers. With this information, users can easily select and keep track of limited regions of the calibration table, even from the command line, by minimizing the number of iterations.
The member function definitions are:
ca.antenna(name=True) - This member function returns the antenna numbers or names as a python list of strings.
ca.calname() - This member function returns the new format calibration table name as a python string.
ca.feed() - This member function returns the feed names as a python list of strings ('X', 'Y' for linear; 'R', 'L' for circular; 'S' for "scalar", when the calibration solutions are performed simultaneously for both polarizations). If the basis is unknown, then the basis functions are '1' and '2'. This kludge was added to handle incomplete calibration tables.
ca.field(name=True) - This member function returns the field numbers or names as a python list of strings.
ca.freq() - This member function returns the frequencies in the table as a python dictionary (the keys are the spectral window numbers and the elements are numpy float arrays containing the frequencies).
ca.msname() - This member function returns the parent measurement set name as a python string.
ca.numantenna() - This member function returns the number of antennas as a python integer.
ca.numchannel() - This member function returns the number of channels for each spectral window as a list of python integers.
ca.numfeed() - This member function returns the number of feeds as a python integer.
ca.numfield() - This member function returns the number of fields as a python integer.
ca.numspw() - This member function returns the number of spectral windows as a python integer.
ca.numtime() - This member function returns the number of times as a python integer.
ca.partype() - This member function returns the parameter column type ('Float' or 'Complex').
ca.polbasis() - This member function returns the polarization basis ('L' for linear or 'C' for circular). If the basis is unknown, 'U' is returned. This kludge was added to handle incomplete calibration tables.
ca.spw(name=True) - This member function returns the spectral window numbers or names as a python list of strings.

ca.time() - This member function returns the times as a python list of floats (in units of MJD seconds). In the future, date strings will be available.
ca.viscal() - This member function returns the type of new formation calibration table as a python string. For example 'B' is a bandpass table, 'G' is a gain table, etc.

Process Member Functions

The process member functions process data. As of CASA 3.4, there are two: get() and fit(). The get() member function iterates through the calibration table and returns the selected data. The fit() member function does the same as the get() member function and returns the fits as well. Tables with complex parameters are converted to either amplitudes or phases.
The get() and fit() member functions employ two levels of iteration. The first level involves field, antenna 1, and antenna2 (slowest to fastest). The data for each first-level iteration are placed in a cube whose dimensions are feed x frequency x time. Two of these dimensions represent the second level of iteration. The feed axis is always an iteration axis, and the user can select either frequency or time as the other one.
In addition to providing a logical way of getting data, this two-level iteration scheme also allows users to fit along the non-iteration axis. For example, a fit can be peformed along the frequency axis for each iteration of the selected field, antenna 1, antenna 2, feed, and time.

Inputs

As mentioned above, the selection syntax originally designed for measurement sets has been implemented in the ca tool. It is available for feed, antenna 1 and 2, and spectral window with channel. For more information, consult the selection documentation. Their argument lists are:
ca.get(field='', antenna='', timerange=[], spw='', feed='', axis='TIME', ap='AMPLITUDE', norm=True, unwrap=True, jumpmax=0.0)
ca.fit(field='', antenna='', timerange=[], spw='', feed='', axis='TIME', ap='AMPLITUDE', norm=True, unwrap=True, jumpmax=0.0, order='AVERAGE', type='LSQ', weight=False)
'*' is equivalent to ''. Both numbers and names can be used for field, antenna, and spw. Names have not been implemented in present EVLA and ALMA datasets for some quantities. Check if they are available using the introspective methods.
The least-squares fit is quite standard. The robust fit, which minimizes the effects of outliers, is experimental. Robust fits are simple to compute, but they don't provide parameter variances and covariances. To minimize outliers and obtain (co)variances, the following algorithm is used:
- Calculate the least-squares fit.
- Using the fit parameters from the least squares fit as starting values, perform

the robust fit (which is essentially a zero-finding algorithm).
- Flag all outliers with residuals greater than 5 times the mean deviation. These flags are actually returned, so they can be applied elsewhere.
- Recalculate the least-squares fit without the outliers.
Arguments for get() and fit():
field = A comma-delimited string or a python list of strings containing the fields. E.g., field = '0,1'. The default is '' (all fields).
antenna = A comma- and semi-colon- delimited string containing the antenna 1s and antenna 2s. E.g, antenna = '3,4,5'. The default is '' (all antenna 1s and antenna 2s).
timerange = A python list of floats of length two containing the start and stop times in MJD seconds. Date strings will be implemented in a future release when they are implemented in the selection C++ code. E.g., timerange = [456123.0,456456.0]. The default is [min MJD, max MJD]. For convenience, the MJD times can be obtained from the time() instrospective method.
spw = A comma- and semi-column- delimited string containing the spectral window and channel selection. E.g., spw = '0:4~20;25~59,2:10~30,6'. The default is '' (all spectral windows and channels).
feed = A comma-delimited string or python list of strings containing the feed names ('X', 'Y', 'R', 'L', or 'S' [scalar]). E.g., feed='X,Y'. The default is '' (all feeds).
axis = A python string containing the user-defined iteration axis ('TIME' or 'FREQ'). E.g., axis='FREQ'. The default is 'TIME' (the frequency axis is a non-iteration axis).
ap = A python string containing the amplitude/phase selection ('AMPLITUDE' or 'PHASE'). E.g., ap = 'PHASE'. The default is 'AMPLITUDE'. It is ignored if the parameters in the calibration table are real.
norm = A python boolean which determines whether amplitudes are normalized for each iteration. E.g., norm = False. The default is True. It is ignored if the parameters in the calibration table are real or ap = 'PHASE'.
unwrap = A python boolean which determines whether phases are unwrapped for each iteration. E.g., unwrap = False. The default is True. It is ignored if the parameters in the calibration table are real or ap = 'AMPLITUDE'.
jumpmax = A python float which determines the maximum phase jump near +/- PI before unwrapping is performed. E.g., jumpmax = 0.1. The default is 0.0. It is ignored if the parameters in the calibration table are real or ap = 'AMPLITUDE'. If the non-iteration axis is frequency:
- if jumpmax == 0.0, use fringe fitting (only available when the non-iteration axis is time).
- if jumpmax != 0.0, use simple unwrapping (same algorithm as used when the non-iteration axis is time or frequency).
Arguments for fit() only:
order = A python string containing the fit order ('AVERAGE', 'LINEAR', or 'QUADRATIC'). E.g., order = 'LINEAR'. The default is 'AVERAGE'. 'QUADRATIC' is not available when the fit type is 'ROBUST'.

type = A python string containing the fit type ('LSQ' or 'ROBUST'). E.g., type = 'ROBUST'. The default is 'LSQ'. The robust fit, which minimizes the effects of outliers, is experimental. Robust fits are simple to compute, but they don't provide parameter variances and covariances. To minimize outliers and obtain (co)variances, the following algorithm is used:
- Calculate the least-squares fit.
- Using the fit parameters from the least squares fit as starting values, perform the robust fit (which is essentially a zero-finding algorithm).
- Flag all outliers with residuals greater than 5 times the mean deviation. These flags are actually returned, so they can be applied elsewhere.
- Recalculate the least-squares fit without the outliers.
weight = A python boolean which determines whether weights are applied. E.g., weight = True. The default is False.

Outputs

The get() and fit() member function return dictionaries of dictionaries. They both return this information (the '#' represents the iteration number):
['#']['field'] = The python string containing the field number.
['#']['antenna1'] = The python string containing the antenna 1 number.
['#']['antenna2'] = The python string containing the antenna 2 number.
['#']['feed'] = A python string containing the feed.
['#']['value'] = The numpy float array containing the parameters (either along the time or frequency axis) from the new format calibration table (if the table contains complex numbers, these numbers are either amplitudes or phases).
['#']['valueErr'] = The numpy float array containing the parameter errors (either along the time or frequency axis) from the new format calibration table (if the table contains complex parameters, these numbers are either amplitude or phase errors).
['#']['flag'] = The numpy boolean array containing the parameter flags.
['#']['abscissa'] = The python string containing the name of the non-iteration axis ('frequency' or 'time').
['#']['frequency'] = The numpy float array containing the frequencies. If the frequency axis is not an iteration axis, the frequencies correspond to the values, value errors, and flags. If the frequency axis is an iteration axis, this array has only one value.
['#']['time'] = The numpy float array containing the times. If the time axis is not an iteration axis, the times correspond to the values, value errors, and flags. If the time axis is an iteration axis, this array has only one value.
['#']['rap'] = The python string containing 'REAL', 'AMPLITUDE', or 'PHASE', describing the values and their errors.
['#']['norm'] = The python boolean determining whether the amplitudes are normalized per iteration or not. It is not present for 'REAL' or 'PHASE' data.
['#']['unwrap'] = The python boolean determining whether the phases are unwrapped per iteration or not. It is not present for 'REAL' or 'AMPLITUDE' data.

['#']['jumpmax'] = The python float containing the maximum phase jump near +/- PI before unwrapping is performed. It is not present for 'REAL' or 'AMPLITUDE' data. If the non-iteration axis is 'frequency':
- if jumpmax == 0.0, fringe fitting was used (only available when the non-iteration axis is time).
- if jumpmax != 0.0, simple unwrapping was unused (same algorithm as used when the non-iteration axis is time or frequency).
In addition to these entries, the fit() member function returns these:
['#']['order'] = The python string describing the fit order ('AVERAGE', 'LINEAR', or 'QUADRATIC'). 'QUADRATIC' is not available for 'ROBUST' fitting.
['#']['type'] = The python string containing the fit type ('LSQ' or 'ROBUST').
['#']['weight'] = The python boolean determining whether the fit was weighted or not.
['#']['validFit'] = The python boolean telling whether the fit was valid or not.
['#']['pars'] = The numpy float array containing the fit parameters.
['#']['vars'] = The numpy float array containing the fit parameter variances.
['#']['covars'] = The numpy float array containing the fit parameter covariances (par0-par1, par0-par2, ..., par1-par2).
['#']['redChi2'] = The python float containing the reduced chi2 (set to 1.0 for unweighted fits).
['#']['model'] = The numpy float array containing the model versus the abscissae.
['#']['res'] = The numpy float array containing the fit residuals versus the absicissae.
['#']['resMean'] = The python float containing the mean of the residuals.
['#']['resVar'] = The python float containing the variance of the residuals.

Methods

calanalysis	Construct a calibration analysis tool.
open	Open a calibration table.
close	Close a calibration table.
calname	Return the calibration table name.
msname	Return the name of the MS that created this calibration table.
viscal	Return the type of calibration table ('B', 'G', 'T', etc.).
partype	Return the parameter column type in the calibration table ('Complex' or 'Float').
polbasis	Return the polarization basis in the calibration table ('L' for linear or 'C' for circular).
numfield	Return the number of fields in the calibration table.
field	Return the fields in the calibration table.
numantenna	Return the number of antennas in the calibration table.
numantenna1	Return the number of antenna 1s in the calibration table.
numantenna2	Return the number of antenna 2s in the calibration table.
antenna	Return the antennas in the calibration table.
antenna1	Return the antenna 1s in the calibration table.

antenna2	Return the antenna 2s in the calibration table.
numfeed	Return the number of feeds in the calibration table.
feed	Return the feeds in the calibration table.
numtime	Return the number of times in the calibration table.
time	Return the times (in MJD seconds) in the calibration table.
numspw	Return the number of spectral windows in the calibration table.
spw	Return the spectral windows in the calibration table.
numchannel	Return the number of channels per spectral window in the calibration table.
freq	Return the frequencies per spectral window in the calibration table.
get	Return the calibration data.
fit	Return the calibration data and fits along the non-iteration axis.

calanalysis.calanalysis.html

calanalysis.calanalysis - Function

2.1.2 Construct a calibration analysis tool.

Description

Construct a calibration analysis tool.

Arguments

Returns
calanalysisobject

---

calanalysis.open.html

calanalysis.open - Function

2.1.2 Open a calibration table.

Description

This member function opens a calibration table.

Arguments

Inputs		
caltable	Python string containing the calibration table name.	
	allowed:	string
	Default:	

Returns

bool

Example

```
ca.open( '<caltable name>' )
```

---

calanalysis.close.html

calanalysis.close - Function

2.1.2 Close a calibration table.

Description

This member function closes a calibration table.

Arguments

Returns
bool

Example

```
ca.close()
```

---

calanalysis.calname.html

calanalysis.calname - Function

2.1.2 Return the calibration table name.

Description

This member function returns calibration table name.

Arguments

Returns
string

Example

```
caltable = ca.calname()
```

---

calanalysis.msname.html

calanalysis.msname - Function

2.1.2 Return the name of the MS that created this calibration table.

Description

This member function returns the name of the MS that created this calibration table.

Arguments

Returns
string

Example

```
msname = ca.msname()
```

---

calanalysis.viscal.html

calanalysis.viscal - Function

2.1.2 Return the type of calibration table ('B', 'G', 'T', etc.).

Description

This member function returns the type of calibration table ('B', 'G', 'T', etc.).

Arguments

Returns
string

Example

```
viscal = ca.viscal()
```

---

calanalysis.partype.html

calanalysis.partype - Function

2.1.2 Return the parameter column type in the calibration table ('Complex' or 'Float').

Description

This member function returns the parameter column type in the calibration table ('Complex' or 'Float').

Arguments

Returns
string

Example

```
partype = ca.partype()
```

---

calanalysis.polbasis.html

calanalysis.polbasis - Function

2.1.2 Return the polarization basis in the calibration table ('L' for linear or 'C' for circular).

Description

This member function returns the polarization basis in the calibration table ('L' for linear or 'C' for circular).

Arguments

Returns
string

Example

```
polbasis = ca.polbasis()
```

---

calanalysis.numfield.html

calanalysis.numfield - Function

2.1.2 Return the number of fields in the calibration table.

Description

This member function returns the number of fields in the calibration table.

Arguments

Returns
int

Example

```
numfield = ca.numfield()
```

---

calanalysis.field.html

calanalysis.field - Function

2.1.2 Return the fields in the calibration table.

Description

This member function returns the fields in the calibration table.

Arguments

Inputs	
name	The python boolean which determines whether field names (True) or field numbers (False) are returned. allowed: boolean Default: true

Returns
stringArray

Example

```
field = ca.field()
```

---

calanalysis.numantenna.html

calanalysis.numantenna - Function

2.1.2 Return the number of antennas in the calibration table.

Description

This member function returns the number of antennas in the calibration table.

Arguments

Returns
int

Example

```
numantenna = ca.numantenna()
```

---

calanalysis.numantenna1.html

calanalysis.numantenna1 - Function

2.1.2 Return the number of antenna 1s in the calibration table.

Description

This member function returns the number of antenna 1s in the calibration table.

Arguments

Returns
int

Example

```
numantenna1 = ca.numantenna1()
```

---

calanalysis.numantenna2.html

calanalysis.numantenna2 - Function

2.1.2 Return the number of antenna 2s in the calibration table.

Description

This member function returns the number of antenna 2s in the calibration table.

Arguments

Returns
int

Example

```
numantenna2 = ca.numantenna2()
```

---

calanalysis.antenna.html

calanalysis.antenna - Function

2.1.2 Return the antennas in the calibration table.

Description

This member function returns the antennas in the calibration table.

Arguments

Inputs	
name	The python boolean which determines whether antenna names (True) or antenna numbers (False) are returned. allowed: boolean Default: true

Returns

stringArray

Example

```
antenna = ca.antenna()
```

---

calanalysis.antenna1.html

calanalysis.antenna1 - Function

2.1.2 Return the antenna 1s in the calibration table.

Description

This member function returns the antenna 1s in the calibration table.

Arguments

Inputs		
name	The python boolean which determines whether antenna 1 names (True) or antenna 1 numbers (False) are returned.	
	allowed:	boolean
	Default:	true

Returns

stringArray

Example

```
antenna1 = ca.antenna1()
```

---

calanalysis.antenna2.html

calanalysis.antenna2 - Function

2.1.2 Return the antenna 2s in the calibration table.

Description

This member function returns the antenna 2s in the calibration table.

Arguments

Inputs		
name	The python boolean which determines whether antenna 2 names (True) or antenna 2 numbers (False) are returned.	
	allowed:	boolean
	Default:	true

Returns

stringArray

Example

```
antenna2 = ca.antenna2()
```

---

calanalysis.numfeed.html

calanalysis.numfeed - Function

2.1.2 Return the number of feeds in the calibration table.

Description

This member function returns the number of feeds in the calibration table.

Arguments

Returns
int

Example

```
numfeed = ca.numfeed()
```

---

calanalysis.feed.html

calanalysis.feed - Function

2.1.2 Return the feeds in the calibration table.

Description

This member function returns the feeds in the calibration table.

Arguments

Returns
stringArray

Example

```
feed = ca.feed()
```

---

calanalysis.numtime.html

calanalysis.numtime - Function

2.1.2 Return the number of times in the calibration table.

Description

This member function returns the number of times in the calibration table.

Arguments

Returns
int

Example

```
numtime = ca.numtime()
```

---

calanalysis.time - Function

2.1.2 Return the times (in MJD seconds) in the calibration table.

Description

This member function returns the times (in MJD seconds) in the calibration table.

Arguments

Returns
doubleArray

Example

```
time = ca.time()
```

---

calanalysis.numspw.html

calanalysis.numspw - Function

2.1.2 Return the number of spectral windows in the calibration table.

Description

This member function returns the number of spectral windows in the calibration table.

Arguments

Returns
int

Example

```
numspw = ca.numspw()
```

---

calanalysis.spw.html

calanalysis.spw - Function

2.1.2 Return the spectral windows in the calibration table.

Description

This member function returns the spectral windows in the calibration table.

Arguments

Inputs	
name	The python boolean which determines whether spectral window names (True) or spectral window numbers (False) are returned. allowed: boolean Default: true

Returns

stringArray

Example

```
spw = ca.spw()
```

---

calanalysis.numchannel.html

calanalysis.numchannel - Function

2.1.2 Return the number of channels per spectral window in the calibration table.

Description

This member function returns the number of channels per spectral window in the calibration table.

Arguments

Returns
intArray

Example

```
numChannel = ca.numchannel()
```

---

calanalysis.freq.html

calanalysis.freq - Function

2.1.2 Return the frequencies per spectral window in the calibration table.

Description

This member function returns the frequencies per spectral window in the calibration table.

Arguments

Returns
record

Example

```
freq = ca.freq()
```

---

calanalysis.get.html

calanalysis.get - Function

2.1.2 Return the calibration data.

Description

This member function returns the calibration data.

Arguments

Inputs	
field	The python comma-delimited string or list of strings containing the field names or numbers. The default is "" (all fields). allowed: variant Default:
antenna	The python comma-delimited string or list of strings containing the antenna 1s and antenna 2s. The default is "" (all antenna 1s and antenna 2s). allowed: variant Default:
timerange	The python list of floats of length two containing the start and stop times (in MJD seconds). The default is [] (the minimum start time and the maximum stop time). allowed: variant Default:
spw	The python comma-delimited string containing the spectral window names and numbers along with their channel numbers. The default is "" (all spectral windows and channels). allowed: variant Default:
feed	The python comma-delimited string or list of strings containing the feeds. The default is "" (all feeds). allowed: variant Default:
axis	The python string containing the user-specified iteration axis. The allowed values are "TIME" and "FREQ". The default is "" ("FREQ"). allowed: string Default: TIME
ap	The python string which determines whether complex gains are converted to amplitudes or phases. The allowed values are "AMPLITUDE" and "PHASE". The default is "" ("AMPLITUDE"). This parameter is ignored when the "gain" values in the calibration table are real. allowed: string Default: AMPLITUDE
norm	The python boolean which determines whether the amplitudes are normalized along each non-iteration axis. The default is False. This parameter is ignored when the "gain" values in the calibration table are real or ap="PHASE". allowed: boolean Default: false
unwrap	The python boolean which determines whether the phases are unwrapped along each non-iteration axis. The default is False. This parameter is ignored when the "gain" values in the calibration table are real or ap="AMPLITUDE". allowed: boolean Default: false
jumpmax	The python float which determines the maximum phase jump near +/- PI before unwrapping is performed. E.g., jumpmax = 0.1. The default is 0.0. It is ignored if the

Returns
record

Example

```
# All data limited only by the spectral window and channel input
data = ca.get( spw="0:4~15,1,2:10~20" )
```

---

calanalysis.fit.html

calanalysis.fit - Function

2.1.2 Return the calibration data and fits along the non-iteration axis.

Description

This member function returns the calibration data and fits along the non-iteration axis.

Arguments

Inputs	
field	The python comma-delimited string or list of strings containing the field names or numbers. The default is "" (all fields). allowed: variant Default:
antenna	The python comma-delimited string or list of strings containing the antenna 1s and antenna 2s. The default is "" (all antenna 1s and antenna 2s). allowed: variant Default:
timerange	The python list of floats of length two containing the start and stop times (in MJD seconds). The default is [] (the minimum start time and the maximum stop time). allowed: variant Default:
spw	The python comma-delimited string containing the spectral window names and numbers along with their channel numbers. The default is "" (all spectral windows and channels). allowed: variant Default:
feed	The python comma-delimited string or list of strings containing the feeds. The default is "" (all feeds). allowed: variant Default:
axis	The python string containing the user-specified iteration axis. The allowed values are "TIME" and "FREQ". The default is "" ("FREQ"). allowed: string Default: TIME
ap	The python string which determines whether complex gains are converted to amplitudes or phases. The allowed values are "AMPLITUDE" and "PHASE". The default is "" ("AMPLITUDE"). This parameter is ignored when the "gain" values in the calibration table are real. allowed: string Default: AMPLITUDE
norm	The python boolean which determines whether the amplitudes are normalized along each non-iteration axis. The default is False. This parameter is ignored when the "gain" values in the calibration table are real or ap="PHASE". allowed: boolean Default: false
unwrap	The python boolean which determines whether the phases are unwrapped along each non-iteration axis. The default is False. This parameter is ignored when the "gain" values in the calibration table are real or ap="AMPLITUDE". allowed: boolean Default: false
jumpmax	The python float which determines the maximum phase jump near +/- PI before unwrapping is performed. E.g., jumpmax = 0.1. The default is 0.0. It is ignored if the

Returns
record

Example

```
# All data limited only by the spectral window and channel input.  The fit order
# is linear.
data_fit = ca.fit( spw="0:4~15,1,2:10~20", order="LINEAR" )
```

---

agentflagger-Module.html

2.2 agentflagger - Module

Module for flagging of synthesis data
include agentflagger

Description The `agentflagger` module provides synthesis flagging capabilities within CASA. The primary purpose of this module is to flag data inside a MeasurementSet.

agentflagger-Tool.html

2.2.1 agentflagger - Tool

Tool for manual and automated flagging
Requires:

Synopsis

Description

The `agentflagger` tool performs manual as well as automatic synthesis flagging operations within casapy. The `agentflagger` tool can operate on one measurement set at a time.

Open the Measurement Set or Calibration Table and Attach it to the Tool
The first thing to do is to open the MS or calibration table and attach it to the agentflagger tool. Use the af.open method, which requires the MS name and optionally the time interval, over which to buffer data before running the algorithm. The time interval is set by default to 0.0, which means a 'scan' length. The 'ntime' parameter is important for the modes tfcrop, rflag and extend.

```
af.open('uid_X002.ms')
```

Select the Data
Once the MS is open, the next step is to select the data. This step will use the MS selection tool to select the portion of the MS given by the parameters. There are two ways of selecting the data:
1) Create a Python dictionary which internally will be transformed into a record containing the selection parameters.

```
# Select the whole MS.
af.selectdata()

Select a portion of the MS using a dictionary.
myrecord={}
myrecord['scan']='1~3'
myrecord['spw']='0:1~10'
af.selectdata(myrecord)
```

2) Parse the parameter names directly to the function.

```
af.selectdata(scan='1~3', spw='0:1~10')
```

Parse the Parameters for the Flagging Mode(s)
Each flagging mode is called an agent. The available agents are: manual, clip, quack, shadow, elevation, tfcrop, rflag, extend, unflag and summary. Each one of these agents may or may not take configuration parameters and data selection parameters. Once the desired flagging modes are chosen, it is time to give the configuration parameters to the tool. Ommited parameters will take default values as defined in each agent. There are two ways of parsing the agent's parameters.
1) Using the general method af.parseagentparameters().

```
Construct a dictionary with the parameters for each agent. Each agent's
parameters should go to a different 'key' of the dictionary. Example:

# Create a shadow agent:
myagents = {}
myagents['mode'] = 'shadow'
af.parseagentparameters(myagents)

# Add a summary agent to the list.
myagents = {}
myagents['mode'] = 'summary
myagents['spwchan] = True
af.parseagentparameters(myagents)

# Add a manual agent to the same internal list of agents.
myagents = {}
myagents['mode'] = 'manual'
myagents['scan'] = '1~3,18~20'
af.parseagentparameters(myagents)

# Add a clip agent to flag the zero-value data.
myagents = {}
myagents['mode'] = 'clip'
myagents['clipzeros'] = True
af.parseagentparameters(myagents)

# Add another summary agent to the list.
myagents = {}
myagents['mode'] = 'summary
myagents['spwchan] = True
af.parseagentparameters(myagents)
```

2) The other way to parse agent's parameters is to use the convenience functions. The above example would become:

```
# Create a shadow agent:
af.parseshadowparameters()

# Add a summary agent to the list.
af.parsesummaryparameters(spwchan=True)

# Add a manual agent to the same internal list of agents.
af.parsemanualparameters(scan='1~3,18~20')

# Add a clip agent to flag the zero-value data.
af.parseclipparameters(clipzeros=True)

# Add another summary agent to the list.
af.parsesummaryparameters(spwchan=True)
```

Initialize the Agents

The above step create a list of the agents that the tool will use to process the data. This step will check several parameters and apply constraints. It will set the iteration approach to
COMBINE_SCANS_MAP_ANTENNA_PAIRS_ONLY if the agent is either tfcrop or extend and combinescans is set to True. Otherwise it will set it to COMPLETE_SCAN_MAP_ANTENNA_PAIRS_ONLY.
If the list contains agents that set ntime more than once, this method will get the maximum value of ntime and use it for all agents.
If a tfcrop agent is present, this method will create one agent per each polarization available, if correlation is set to ALL.
In the same way, if an agent tfcrop, rflag or clip is present, the asyncio mechanism will be switched on.

```
af.init()
```

Run the tool

Run the tool to apply or unapply the flags. The run method takes two parameters, writeflags and sequential. The parameter writeflags controls whether to write the flags or not to the MS. By default it is set to True. The sequential parameter tells to apply/unapply the flags in parallel or not. By default it is set to True, which means that the agents will run in sequential. The run method gathers several reports, depending on wich agents are run. The display and summary agents produce reports that can be retrieved from calling the run method. The reports are returned via a Python dictionary.

```
myreports = af.run(writeflags=True)
```

The dictionary returned in 'myreports' will contain four reports from the two summary agents that were added previously. The first report is the normal summary for each selection parameter. The second report gives the antenna positions for plotting.

Destroy the tool
Do not forget to destroy and close the tool at the end.

```
af.done()
```

Methods

agentflagger	Construct a flag tool
done	Destroy the flag tool
open	Open the MS or a calibration table and attach it to the tool.
selectdata	Select the data based on the given parameters. For unspecified parameters, the
parseagentparameters	Parse the parameters for the agent (flagging mode).
init	Initialize the agents
run	Execute a list of flagging agents
getflagversionlist	Print out a list of saved flag_versions.
printflagselection	Print out a list of current flag selections.
saveflagversion	Save current flags with a version name.
restoreflagversion	Restore flags from a saved flag_version. versionname : name of flag version to re
deleteflagversion	Delete a saved flag_version.
parsemanualparameters	Parse data selection parameters and specific parameters for the manual mode. I
parseclipparameters	Parse data selection parameters and specific parameters for the clip mode. Data
parsequackparameters	Parse data selection parameters and specific parameters for the quack mode. Da
parseelevationparameters	Parse data selection parameters and specific parameters for the elevation mode.
parsetfcropparameters	Parse data selection parameters and specific parameters for the time and freque
parseextendparameters	Parse data selection parameters and specific parameters for the extend mode. D
parsesummaryparameters	Parse data selection parameters and specific parameters for the summary mode.

agentflagger.agentflagger.html

agentflagger.agentflagger - Function

2.2.1 Construct a flag tool

Description

Create a `agentflagger` tool, and initialize some variables.

Arguments

Inputs

Returns

unknown

Example

```
af.agentflagger()
```

---

agentflagger.done.html

agentflagger.done - Function

2.2.1 Destroy the flag tool

Arguments

Returns
bool

Example

```
af.done()
```

---

agentflagger.open.html

agentflagger.open - Function

2.2.1 Open the MS or a calibration table and attach it to the tool.

Arguments

Inputs		
msname	Measurement set or calibration table to be processed.	
	Default:	
	allowed:	string
	Default:	
ntime	Time interval. If not given, the default will be used.	
	Default:	
	allowed:	double
	Default:	0.0

Returns
bool

Example

```
af.open(msname,ntime)
```

---

agentflagger.selectdata.html

agentflagger.selectdata - Function

2.2.1 Select the data based on the given parameters. For unspecified parameters, the full data range is assumed. All data selection parameters follow the MS Selection syntax.

Arguments

Inputs	
config	The record (dictionary) config may be given or not. If it is not given, and no specific parameter is given either, the whole MS will be selected. The record may contain any data selection parameters supported by MS Selection such as: allowed: record Default:
field	Field indices or source names : example : '2,3C48' allowed: string Default:
spw	Spectral Window Indices or names : example : '1,2' allowed: string Default:
array	Array Indices or names : example : 'VLAA' allowed: string Default:
feed	Feed index or name : example : '1,2' (not supported yet) allowed: string Default:
scan	Scan number : example : '1,2,3' allowed: string Default:
antenna	Baseline number : example : '2,3,4,5' allowed: string Default:
uvrange	UV-distance range, with a unit : example : '2.0-3000.0 m' allowed: string Default:
timerange	Time range, as MJDs or date strings : example : 'xx.x.x.x.x~yy.y.y.y.y' allowed: string Default:
correlation	Correlations/polarizations : example : 'RR,LL,RL,LR,XX,YY,XY,YX,Sol1' allowed: string Default:
intent	Scan intent : example : '*CAL*, *BAND*' allowed: string Default:
observation	Observation Id : example : '2~4' allowed: string Default:

Returns
bool

Example

```
Select the whole MS
af.selectdata()

Select a portion of the MS
myrecord={}
myrecord['scan']='1~3'
myrecord['spw']='0:1~10'
af.selectdata(myrecord)

Another way to select a portion of the MS
af.selectdata(scan='3~5', spw='0')
```

---

agentflagger.parseagentparameters.html

agentflagger.parseagentparameters - Function

2.2.1 Parse the parameters for the agent (flagging mode).

Description

The specific data selection parameters for the agent (flagging mode) are parsed. These parameters are the data selection and mode-specific parameters. See the example below:

Arguments

Inputs	
aparams	It takes a record (dictionary) with the specific parameters for the flagging mode. The record may contain any data selection parameters supported by MS Selection, as well as mode-specific parameters such as:(1) array,feed,scan,field,spw,intent,correlation,antenna,uvrange,observation (2) mode (which can be: manual,clip,quack,shadow,elevation,tfcrop,extendflags,unflag or summary) For flagging mode=clip, the parameters are: expression, datacolumn, clipminmax, etc. See the documentation of the task flagdata for all the available parameters for each mode.(3) apply: default is true (true for flagging and false for unflagging) Example:myrecord=myrecord['mode']='clip'myrecord['scan']='1~3'myrecord['clipminmax' allowed: record Default:

Returns

bool

Example

```
myrecord={}
```

```
myrecord['mode']='clip'
myrecord['scan']='1~3'
myrecord['clipminmax']=[0.02,0.3]
myrecord['apply']=True
af.parseagentparameters(myrecord)
```

---

agentflagger.init.html

agentflagger.init - Function

2.2.1 Initialize the agents

Description

This method will initialize the agents and create a list of agents with their specific parameters. It takes no parameters.

Arguments

Inputs

Returns

bool

Example

```
af.init()
```

---

agentflagger.run.html

agentflagger.run - Function

2.2.1 Execute a list of flagging agents

Description

Execute a list of flagging agents and write or not to the MS/cal table. The parameter writeflags controls whether or not to write to the MS.

Arguments

Inputs		
writeflags	Write flags to MS	
	allowed:	bool
	Default:	true
sequential	Run the agents in the order they are inserted in the list or not. Default is True to run in the original order.	
	allowed:	bool
	Default:	true

Returns

record

Example

```
af.run()
```

---

agentflagger.getflagversionlist.html

agentflagger.getflagversionlist - Function

2.2.1 Print out a list of saved flag_versions.

Description

Print out the list of flag versions in the MS, unless the parameter printflags=False. The list of names is returned.

Arguments

Inputs		
printflags	Print flagversions in logger?	
	allowed:	bool
	Default:	true

Returns

stringArray

---

agentflagger.printflagselection.html

agentflagger.printflagselection - Function

2.2.1 Print out a list of current flag selections.

Description

Print out a list of current flag selections.

Arguments

Inputs

Returns

bool

---

agentflagger.saveflagversion.html

agentflagger.saveflagversion - Function

2.2.1 Save current flags with a version name.

Description

Arguments

Inputs		
versionname	Version name	
	allowed:	string
	Default:	
comment	Comment for this flag table	
	allowed:	string
	Default:	
merge	merge type	
	allowed:	string
	Default:	

Returns
bool

---

agentflagger.restoreflagversion.html

agentflagger.restoreflagversion - Function

2.2.1 Restore flags from a saved flag_version. versionname : name of flag version to restore to main table merge : Type of operation to perform during restoration. merge = replace : replaces the main table flags. merge = and : logical AND with main table flags merge = or : logical OR with main table flags Default : replace.

Description

Arguments

Inputs		
versionname	Version name	
	allowed:	stringArray
	Default:	
merge	merge type	
	allowed:	string
	Default:	

Returns
bool

---

agentflagger.deleteflagversion.html

agentflagger.deleteflagversion - Function

2.2.1 Delete a saved flag_version.

Description

Arguments

Inputs		
versionname	Version name	
	allowed:	stringArray
	Default:	

Returns
bool

---

agentflagger.parsemanualparameters.html

agentflagger.parsemanualparameters - Function

2.2.1 Parse data selection parameters and specific parameters for the manual mode. Data selection follows the MS Selection syntax.

Arguments

Inputs	
field	Field indices or source names. Example: '2,3C48' allowed: string Default:
spw	Spectral Window Indices or names. Example: '1,2' allowed: string Default:
array	Array Indices or names. Example: 'VLAA' allowed: string Default:
feed	Feed index or name. Eexample: '1,2' (not supported yet) allowed: string Default:
scan	Scan number. Example: '1,2,3' allowed: string Default:
antenna	Baseline number. Example: '2,3,4,5,132' allowed: string Default:
uvrange	UV-distance range, with a unit. Example: '2.0-3000.0 m' allowed: string Default:
time	Time range, as MJDs or date strings. Example: 'xx.x.x.x.x∼yy.y.y.y.y' allowed: string Default:
correlation	Correlations/polarizations. Example: 'RR,LL,RL,LR,XX,YY,XY,YX' allowed: string Default:
intent	Scan intent. Example: '*CAL*, *BAND*' allowed: string Default:
observation	Observation Id. Example: '2∼4' allowed: string Default:
autocorr	Parameter to flag only auto-correlations. Default: allowed: bool Default: false
apply	Parameter to flag or unflag the data. Default: allowed: bool Default: true

Returns
bool

Example

```
af.parsemanualparameters(autocorr=True)
```

---

agentflagger.parseclipparameters.html

agentflagger.parseclipparameters - Function

2.2.1 Parse data selection parameters and specific parameters for the clip mode. Data selection follows the MS Selection syntax.

Arguments

Inputs	
field	Field indices or source names. Example: '2,3C48' allowed: string Default:
spw	Spectral Window Indices or names. Example: '1,2' allowed: string Default:
array	Array Indices or names. Example: 'VLAA' allowed: string Default:
feed	Feed index or name. Eexample: '1,2' (not supported yet) allowed: string Default:
scan	Scan number. Example: '1,2,3' allowed: string Default:
antenna	Baseline number. Example: '2,3,4,5' allowed: string Default:
uvrange	UV-distance range, with a unit. Example: '2.0-3000.0 m' allowed: string Default:
time	Time range, as MJDs or date strings. Example: 'xx.x.x.x.x~yy.y.y.y.y' allowed: string Default:
correlation	Correlations/polarizations. Example: 'RR,LL,RL,LR,XX,YY,XY,YX' allowed: string Default:
intent	Scan intent. Example: '*CAL*, *BAND*' allowed: string Default:
observation	Observation Id. Example: '2~4' allowed: string Default:
datacolumn	Data column to use for clipping. Supported columns for cal tables are FPARAM,CPARAM,SNR. Example: 'DATA'. Default: allowed: string Default: DATA
clipminmax	Range to use for clipping. Example: [100.0,200.0] allowed: doubleArray Default:
clipoutside	Clip points outside this range? [True/False]. Default: allowed: bool Default: true
channelavg	Average data over channels before clipping? [True/False]. Default: allowed: bool Default: false
chanbin	Width (bin) of input channels to average to form an output channel. allowed: variant

Returns
bool

Example

```
The correlation parameter can be used with an operator for the clip mode.
The operator should be written only once and it will affect all the polarizations
given in the string. See the example below:

af.parseclipparameters(clipzeros=True, clipminmax=[0.,4.], correlation='ABS_XX,XY')

or for a calibration table:
af.parseclipparameters(clipzeros=True, clipminmax=[0.,4.], correlation='Sol1')
```

---

agentflagger.parsequackparameters.html

agentflagger.parsequackparameters - Function

2.2.1 Parse data selection parameters and specific parameters for the quack mode. Data selection follows the MS Selection syntax.

Arguments

Inputs	
field	Field indices or source names. Example: '2,3C48' allowed: string Default:
spw	Spectral Window Indices or names. Example: '1,2' allowed: string Default:
array	Array Indices or names. Example: 'VLAA' allowed: string Default:
feed	Feed index or name. Eexample: '1,2' (not supported yet) allowed: string Default:
scan	Scan number. Example: '1,2,3' allowed: string Default:
antenna	Baseline number. Example: '2,3,4,5' allowed: string Default:
uvrange	UV-distance range, with a unit. Example: '2.0-3000.0 m' allowed: string Default:
time	Time range, as MJDs or date strings. Example: 'xx.x.x.x.x~yy.y.y.y.y' allowed: string Default:
correlation	Correlations/polarizations. Example: 'RR,LL,RL,LR,XX,YY,XY,YX' allowed: string Default:
intent	Scan intent. Example: '*CAL*, *BAND*' allowed: string Default:
observation	Observation Id. Example: '2~4' allowed: string Default:
quackmode	Quack mode. Default: allowed: string Default: beg
quackinterval	Quack length in seconds. Default: allowed: double Default: 0.0
quackincrement	Flag incrementally in time. Default: allowed: bool Default: false
apply	Parameter to flag or unflag the data. Default: allowed: bool Default: true

Returns
bool

Example

```
af.parsequackparameters(scan='1~3', quackmode='beg', quackinterval=1)
```

---

agentflagger.parseelevationparameters.html

agentflagger.parseelevationparameters - Function

2.2.1 Parse data selection parameters and specific parameters for the elevation mode. Data selection follows the MS Selection syntax.

Arguments

Inputs	
field	Field indices or source names. Example: '2,3C48' allowed: string Default:
spw	Spectral Window Indices or names. Example: '1,2' allowed: string Default:
array	Array Indices or names. Example: 'VLAA' allowed: string Default:
feed	Feed index or name. Eexample: '1,2' (not supported yet) allowed: string Default:
scan	Scan number. Example: '1,2,3' allowed: string Default:
antenna	Baseline number. Example: '2,3,4,5' allowed: string Default:
uvrange	UV-distance range, with a unit. Example: '2.0-3000.0 m' allowed: string Default:
time	Time range, as MJDs or date strings. Example: 'xx.x.x.x.x~yy.y.y.y.y' allowed: string Default:
correlation	Correlations/polarizations. Example: 'RR,LL,RL,LR,XX,YY,XY,YX' allowed: string Default:
intent	Scan intent. Example: '*CAL*, *BAND*' allowed: string Default:
observation	Observation Id. Example: '2~4' allowed: string Default:
lowerlimit	The limiting elevation in degrees. Data obtained at lower antenna elevations will get flagged. Default: allowed: double Default: 0.0
upperlimit	The limiting elevation in degrees. Data obtained at higher antenna elevations will get flagged. Default: allowed: double Default: 90.0
apply	Parameter to flag or unflag the data. Default: allowed: bool Default: true

Returns
bool

Example

```
To unflag, set the apply parameter.
af.parseelevationparameters(upperlimit=50.,lowerlimit=10.0, apply=False)
```

---

agentflagger.parsetfcropparameters - Function

2.2.1 Parse data selection parameters and specific parameters for the time and frequency mode. Data selection follows the MS Selection syntax.

Arguments

Inputs	
field	Field indices or source names. Example: '2,3C48' allowed: string Default:
spw	Spectral Window Indices or names. Example: '1,2' allowed: string Default:
array	Array Indices or names. Example: 'VLAA' allowed: string Default:
feed	Feed index or name. Eexample: '1,2' (not supported yet) allowed: string Default:
scan	Scan number. Example: '1,2,3' allowed: string Default:
antenna	Baseline number. Example: '2,3,4,5' allowed: string Default:
uvrange	UV-distance range, with a unit. Example: '2.0-3000.0 m' allowed: string Default:
time	Time range, as MJDs or date strings. Example: 'xx.x.x.x.x~yy.y.y.y.y' allowed: string Default:
correlation	Correlations/polarizations. Example: 'RR,LL,RL,LR,XX,YY,XY,YX' allowed: string Default:
intent	Scan intent. Example: '*CAL*, *BAND*' allowed: string Default:
observation	Observation Id. Example: '2~4' allowed: string Default:
ntime	Time-range to use for each chunk (in seconds or minutes). Default: allowed: double Default: 0.0
combinescans	Accumulate data across scans depending on the value of ntime. Default: allowed: bool Default: false
datacolumn	Data column to use for clipping. Example: 'DATA'. Default: allowed: string Default: DATA
timecutoff	Flagging thresholds in units of deviation from the fit. Default: allowed: double Default: 4.0
freqcutoff	Flagging thresholds in units of deviation from the fit. Default:

Returns
bool

Example

```
The correlation parameter can be used with an operator for the tfcrop mode.
The operator should be written only once and it will affect all the polarizations
given in the string. Note that if ntime='scan' and combinescans=True, all the scans will
be loaded at once, thus requesting a lot of memory depending on the available spws.
The parameter combinescans should be set to True only when ntime is specified as a
time-interval (not 'scan'). By default, the flags will be extended in time, if
more than 50\% of the timeranges are flagged, 80\% of the channels are flagged and
it will extend to other polarizations too. This is similar to running the extend
mode after running tfcrop on the MS.

af.parsetfcropparameters(spw='9', ntime=10.0, combinescans=True, correlation='ABS_XX,XY',
                         extendflags=True)
```

---

agentflagger.parseextendparameters.html

agentflagger.parseextendparameters - Function

2.2.1 Parse data selection parameters and specific parameters for the extend mode. Data selection follows the MS Selection syntax.

Arguments

Inputs	
field	Field indices or source names. Example: '2,3C48' allowed: string Default:
spw	Spectral Window Indices or names. Example: '1,2' allowed: string Default:
array	Array Indices or names. Example: 'VLAA' allowed: string Default:
feed	Feed index or name. Eexample: '1,2' (not supported yet) allowed: string Default:
scan	Scan number. Example: '1,2,3' allowed: string Default:
antenna	Baseline number. Example: '2,3,4,5' allowed: string Default:
uvrange	UV-distance range, with a unit. Example: '2.0-3000.0 m' allowed: string Default:
time	Time range, as MJDs or date strings. Example: 'xx.x.x.x.x~yy.y.y.y.y' allowed: string Default:
correlation	Correlations/polarizations. Example: 'RR,LL,RL,LR,XX,YY,XY,YX' allowed: string Default:
intent	Scan intent. Example: '*CAL*, *BAND*' allowed: string Default:
observation	Observation Id. Example: '2~4' allowed: string Default:
ntime	Time-range to use for each chunk (in seconds or minutes). Default: allowed: double Default: 0.0
combinescans	Accumulate data across scans.. Default: allowed: bool Default: false
extendpols	If any correlation is flagged, flag all correlations. Default: allowed: bool Default: true
growtime	Flag all 'ntime' integrations if more than X% of the timerange is flagged (0-100). Default: allowed: double Default: 50.0
growfreq	Flag all selected channels if more than X% of the frequency range is flagged(0-100). Default: allowed: double

Returns
bool

Example

```
af.parseextendparameters(extendpols=True)
```

---

agentflagger.parsesummaryparameters.html

agentflagger.parsesummaryparameters - Function

2.2.1 Parse data selection parameters and specific parameters for the summary mode. Data selection follows the MS Selection syntax.

Arguments

Inputs	
field	Field indices or source names. Example: '2,3C48' allowed: string Default:
spw	Spectral Window Indices or names. Example: '1,2' allowed: string Default:
array	Array Indices or names. Example: 'VLAA' allowed: string Default:
feed	Feed index or name. Eexample: '1,2' (not supported yet) allowed: string Default:
scan	Scan number. Example: '1,2,3' allowed: string Default:
antenna	Baseline number. Example: '2,3,4,5' allowed: string Default:
uvrange	UV-distance range, with a unit. Example: '2.0-3000.0 m' allowed: string Default:
time	Time range, as MJDs or date strings. Example: 'xx.x.x.x.x~yy.y.y.y.y' allowed: string Default:
correlation	Correlations/polarizations. Example: 'RR,LL,RL,LR,XX,YY,XY,YX' allowed: string Default:
intent	Scan intent. Example: '*CAL*, *BAND*' allowed: string Default:
observation	Observation Id. Example: '2~4' allowed: string Default:
spwchan	List the number of flags per spw and per channel. Default: allowed: bool Default: false
spwcorr	List the number of flags per spw and per correlation. Default: allowed: bool Default: false
basecnt	List the number of flags per baseline. Default: allowed: bool Default: false
fieldcnt	List the number of flags per field. Default: allowed: bool Default: false
name	Name of this summary report. Default: summary allowed: string Default:

Returns
bool

Example

```
af.parsesummaryparameters(spwchan=True, basecnt=True)
```

---

imager-Module.html

2.3 imager - Module

Module for synthesis and single dish imaging
include imager.g

`imager` provides a unified interface for synthesis and single dish imaging including deconvolution starting from a MeasurementSet.

What `imager` does:

Standard synthesis and single dish imaging `imager` does nearly all types of synthesis and single dish imaging, including dirty images, point spread functions, deconvolution, combination of single dish and synthesis, spectral imaging, polarimetry, wide-field imaging, mosaicing, holography, near-field imaging, tracking moving objects, *etc.*. As a result of this extensive range of capabilities, it can be complicated to use, especially for the more esoteric forms of imaging.

Fine scaled tools Rather than present one operation to process data from visibilities to a restored, deconvolved image, `imager` contains a number of distinct, separate tool functions that allow careful tuning of the processing. For example, the weights used in imaging (the IMAGING_WEIGHT column in the MeasurementSet), can be altered via a number of tool functions (weight, filter) and inspected via a plotting tool function plotweights. Similarly, the deconvolution and restoration steps are separate, allowing user control of each step. It is our intention that other imaging tools may be built on top of imager: see, for example, imagerwizard, which also has the side-benefit that it displays the imager (and other tools) commands as they are executed.

Spectral imaging `imager` can perform either spectral imaging or frequency synthesis (producing either an image with each channel imaged independently or with some or all channels summed together). Channels may be selected in a number of ways, either as channels or as velocities. Also a continuum model image may be subtracted prior to making a cube.

Many different deconvolution algorithms `imager` is rich in deconvolution algorithms, including a number of clean variants, maximum entropy, nonnegative least squares, and the pixon algorithm.

Mixing of deconvolution functions Since the deconvolved images are calculated and kept purely as images (rather than lists of clean components), deconvolution functions may be mixed as desired. Thus, one may use NNLS to deconvolve part of the Stokes I of an image, and then use CLEAN to deconvolve another part of all polarizations in the image. Note that a list of clean components is not available.

Ability to fix model images In a multifield deconvolution, it is possible to specify that some fields are not to be deconvolved, using the `fixed` argument of clean.

Single dish imaging `imager` can process single dish observations much as it does synthesis images. To make images with no deconvolution, use the makeimage function. This allows construction of traditional single dish images and holography images. To deconvolve images, just use the "multifield" deconvolution algorithms in clean and mem. You will want to set the gridmachine in setoptions to 'sd'.

Combination of single dish and synthesis data If the single dish and interferometer data are in the same MeasurementSet, then imager can perform a joint deconvolution using "multifield" deconvolution algorithms in clean and mem. You will want to set the gridmachine in setoptions to 'both'. You can change the relative weighting of synthesis and single dish data by using setsdoptions. If the single dish and synthesis data cannot be combined into one MeasurementSet then you can still use the feather function to combine already deconvolved images.

Multi-field processing `imager` can be run on any number of images, each of which can have any direction for the phase center. All coordinate transformations are done correctly. Using the measures system, these fields may be given moving positions (such as the Sun using `dm.direction('sun')` to specify the phase center) or positions in strange coordinates (such as Supergalactic using *e.g.* `dm.direction('supergal', '0d', '0d')` as well as the more conventional representations (*e.g.* `dm.direction('b1950', '12h26m33.248000', '02d19m43.290000')` specifies the coordinates of the core of 3C273). Note that for some coordinate systems a location must be supplied via the setoptions tool function. For example, one can put an image at a specific azimuth and elevation (*e.g.* `dm.direction('azel',`

`'67.4d', '5.23d'`) at the VLA `imgr.setoptions(location=dm.observatory('VLA'))`. Phase rotation will be automatically calculated to track in azimuth-elevation.

Wide-field imaging `imager` can perform wide-field imaging as needed to overcome the non-coplanar baselines effect for the VLA and other non-coplanar arrays.

Mosaicing `imager` can perform clean-based or mem-based mosaicing of many pointings into one image, using variants of the multi-field algorithms.

Processing of component lists Discrete components (not the same as clean components!) can be represented by componentmodels. A componentlist can hold any number of components. The components are subtracted from the visibility data before construction of an image. For high precision imaging, it is recommended that components be used for bright sources since the Fourier Transform of components avoids the limitations of the gridded transforms.

Joint deconvolution of Stokes IQUV `imager` can produce images of either I alone, or I, V or I, Q, U, V, deconvolving jointly as appropriate. The point spread function is constrained to be the same for all processed polarizations so asymmetric u,v coverage is not allowed.

Production of complex images `imager` can produce dirty or residual images or point spread functions in the original data representation (*e.g.* RR,RL,RL,LL or XX,XY,YX,YY.

Fine control and evaluation of visibility weighting Various tool functions for controlling the visibility weights are available (weight, filter) as well as tool functions for evaluating the effects of the weighting (plotweights, sensitivity, fitpsf). The Briggs algorithm for weighting of visibility data can be used (see weight and Dan Briggs' thesis).

Flexible windowing in the deconvolution Rather than use boxes to limit the region CLEANed, a mask image is used to constrain the region in which flux is allowed. There are various tool functions for making a mask image, including from regions and blc/trc specifications, and via thresholding the Stokes I image. In the Clark Clean, the mask is *soft*: it can vary between 0 and 1. Intermediate values of the mask bias against but do not rule out subtraction of clean components.

Non-Negative Least Squares Deconvolution This algorithm is very effective at producing high dynamic range images of moderately resolved sources (see Dan Briggs' thesis). It works on Stokes I alone so the recommended procedure is to CLEAN I, Q, U, V using clean and then use NNLS to refine the I part of the image using nnls.

Specification of arguments as A measure is a measured quantity with optional units, coordinates and reference frames. These are allowed in a

number of circumstances. The advantage is that the user can specify arguments in very convenient form, and let the measures system do whatever conversion is required. For example:

Cell sizes These can be specified as a quantity (see the measures module).

```
imgr.setimage(cellx='7arcsec', celly='7arcsec')
```

Image center direction This must be specified as a direction (see the measures module).

```
imgr.setimage(phasecenter=dm.direction('j2000', '05h30m', '-30.2deg'))
imgr.setimage(phasecenter=dm.direction('gal', '0deg', '0deg'))
imgr.setimage(phasecenter=dm.direction('mars'))
imgr.setimage(phasecenter=image('myother.image').coordmeasures().direction);
```

Velocities These can be specified as radial velocities.

```
imgr.setimage(start=dm.radialvelocity('25km/s'),
              step=dm.radialvelocity('-500m/s'))
```

Position For construction of images in some coordinate frames (*e.g* azimuth-elevation) the position to be used in processing must be set:

```
imgr.setoptions(location=dm.observatory('ATCA'))
```

More choice in image size Any even image size will work, though to speed the FFT, it is advisable to use a highly composite number (one that has many factors). The advise function will calculate an acceptable number.

Integrated plotting Plots of visibility amplitude, weights (both point-by-point and gridded), uv coverage, and field and spectral window ids are available (plotvis, plotweights, plotuv, plotsummary).

Synchronous or Asynchronous processing Operations that take a substantial amount of time to run can be run in the background either by setting the global variable `dowait:=F` or by setting an argument *e.g.* `imgr.clean(async=T)`. To retrieve a result, use the result tool function of defaultservers with the job number as the argument. For example:

```
- imgr:=imager('ss433.MS')
T
- imgr.setimage(cellx='0.05arcsec', celly='50marcsec', nx=256, ny=256,
  spwid=1:2, fieldid=1, stokes='IV')
T
- imgr.fitpsf()
1
# Wait for it to finish and then ask for the result:
- defaultservers.result(1)
[psf=, bpa=[value=42.7269936, unit=deg], bmin=[value=0.13008301,
unit=arcsec], bmaj=[value=0.159367442, unit=arcsec]]
```

A novel sort-less gridding algorithm The visibility data are not sorted before the gridding step. Instead, a cache of tiles is allocated to hold each baseline as it moves around in the Fourier plane. When a baseline moves off an existing tile, the results are written to disk and the necessary new tile is read in. Since the rotation of baselines in the uv plane is usually quite slow, the hit rate of such a cache is high. The size of the cache is by default set to half the physical memory of the machine, as specified by the aipsrc variable system.resources.memory. This can be overridden by the user, via the setoptions tool function. The cache can be made smaller at the expense of more paging of tiles in and out. The tile size can also be changed but this is seldom needed. This approach is optimal for arrays with small numbers of antennas but can be slow for *e.g.* the VLA. We intend to rectify this in the near future.

Plug-in commands `imager` can be customized by attaching commands using the CASA plug-in system. See the file code/trial/apps/imager/imager_standard.gp for an example of how to attach commands.

Suite of tests `imager` has a suite of tests. A standard test data set and component list can also be created.

imagerwizard The simplemage function is a wizard that performs interactively guided imaging of synthesis data.

dragon The dragon tool performs wide-field imaging using imager.

vpmanager The vpmanager tool manages specification of primary beams for imager.

Near-field imaging *experimental* Images of objects in the near-field of an array can be made. If the distance to the object is specified in setimage, then the extra delay due to the wavefront curvature is corrected in the transforms. Note that some telescopes (*e.g.* VLA) make this correction in the real-time system. This effect is important if the distance to the object is comparable to or less than:

$$\frac{B^2}{\lambda} \tag{2.1}$$

where B is the baseline. Note that the sign of the correction could be in error in this experimental version: try using a negative distance as well as a positive distance.

What `imager` needs:

`imager` operates on a specified MeasurementSet to produce any of a range of different types of image: dirty, point spread function, clean, residual, *etc.* A MeasurementSet is the holder for measurements from a telescope. It is simply

an CASA Table obeying certain conventions as to required and optional contents. The intention is that it should contain all the information needed to reduce synthesis and single dish observations (see CASA Note 191). A UVFITS file can be converted to a MeasurementSet using the fitstoms tool function (a constructor of the ms tool.

`imager` adds some extra columns to the MeasurementSet to store results of processing. The following columns in the MS are particularly important:

DATA The original observed visibilities are in a column called DATA. These are not altered by any processing in CASA.

CORRECTED_DATA During a calibration process, as carried out by *e.g.* calibrater, the visibilities may be corrected for calibration effects. This corrected visibilities are stored in a column CORRECTED_DATA which is created on demand by calibrater and imager. In creating the CORRECTED_DATA column, `imager` will only correct for parallactic angle rotation. This can be controlled using the correct tool function. All imaging performed by imager is from the CORRECTED_DATA column (apart from the tool function makeimage which can also make dirty images from the other visibility columns).

MODEL_DATA During various phases of processing, the visibilities as predicted from some model are required. These model visibilities are stored in a column MODEL_DATA. These are used by the calibrater tool for calibration.

IMAGING_WEIGHT Weighting of data (including natural, uniform and Briggs weighting, and tapering) is accomplished by setting the column IMAGING_WEIGHT appropriately.

Standard tools such as the table module and the ms can be used to access and possibly change these (and all other) columns.

`imager` can handle an initial model in a number of forms: as an image, as a list of images, as a componentmodels:componentlist, or as some combination. Fitting of `componentmodels` is planned but is not currently supported.

`imager` uses a number of scratch files. Following CASA practice, these are placed in the directories specified in the aipsrc variable user.directories.work. Those disks that possess sufficient free disk space are chosen in sequence. So to spread your scratch files over two disks each of which has a directory tcornwel/tmp do *e.g.*

```
user.directories.work:   /bigdisk1/tcornwel/tmp /bigdisk2/tcornwel/tmp
```

How to control imager:

To use `imager`, one has to construct a `imager` tool using a MeasurementSet as an argument, for example:

```
myimager:=imager('3C273XC1.ms')
```

The Glish variable `myimager` then contains the tool functions that may be used to do various operations on the MeasurementSet 3C273XC1.ms. These tool functions can be broken down into those that set `imager` up in some state, and those that actually do some processing. The setup tool functions are:

setimage is a *required* tool function that defines the parameters (size, sampling, phase center, *etc.*) of any image that is to be constructed. If you omit to call setimage prior to any operation that needs these parameters, an error message will result. setimage is passive: nothing happens immediately but subsequent processing is altered.

setdata is an *optional* tool function that selects which data are to be operated on during the processing. This selection can consist of choosing the spectral windows or fields that are to be operated on, or setting channels that are to be operated on in subsequent processing. setdata is active: the selection occurs immediately and is effective for all subsequent operations (until setdata is called again).

setoptions is an *optional* tool function that sets parameters of lesser importance such as gridding parameters, cache sizes. While these affect the processing, usually the default values will suffice. setoptions is passive: nothing happens immediately but subsequent processing is altered.

setbeam is an *optional* tool function that sets the parameters of the synthesized beam to be used in restoring deconvolved images. setbeam is passive: nothing happens immediately but subsequent processing is altered.

setvp is an *optional* tool function that sets the parameters of the voltage pattern model used in mosaicing. setvp is passive: nothing happens immediately but subsequent processing is altered.

setsdoptions is an *optional* tool function that sets the relative scaling and weighting of single dish data versus interferometer data and also other single dish specific parameters like the convolution support when doing single dish imaging.

Thus to understand what imager is doing, one has to remember that at any time, it has a *state* that has been set by using these tool functions. The state may be viewed in one of two ways: either summary can be used to output the current state to the logger, or, in the GUI, the current state of these parameters is displayed and updated following any relevant changes.

All the other tool functions of `imager` are active: something happens immediately. Hence, for example, the weight tool function acts immediately to change the weighting of the selected data. In particular, unlike other packages, it does *not* set the weighting parameters for latter operations. The clean tool function performs a clean deconvolution of an image, reading and writing a model image. Note that operations that require or produce an image usually take an appropriate image name in the argument list. Often if such an image is not given then it is constructed using the image parameters set via setimage and using an appropriate name (*e.g.* a restored image is named from the model image by appending .restored so that 3C273XC1.clean becomes 3C273XC1.clean.restored).

The concept of the *state* of imager bears a little more explanation. The MeasurementSet can potentially contain data for many different fields and spectral windows. One therefore has to have some way of distinguishing which data are to be included in processing. Rather than have each possible tool function (*e.g.* weight, image, clean) take a long list of parameters to determine which data are to be included, `imager` has a setdata tool function that sets `imager` up some that in subsequent processing only the selected data are processed. For example, to select only field id 1 and spectral windows 1 and 2, one would do:

```
myimager.setdata(fieldid=1, spwid=1:2)
```

The state of `imager` also consists of information about the default image settings (set via setimage) and various less important options (set via setoptions).

What `imager` produces:

`imager` reads and writes CASA MeasurementSets and Images. The format of images is 4 dimensional, with the first two being right ascension and declination, the third being polarization and the fourth being frequency. By suitable choice of the input parameters, one can make images of I alone, I, V or I, Q, U, V for one or all channels. The makeimage tool function can also make a complex image of the original polarizations *e.g.* RR, RL, LR, LL. This latter type of image is useful for diagnostic purposes.

Images generated by `imager` may be viewed using viewer tool or retrieved using the images tool, the MeasurementSets may be accessed using the ms tool. More on this in the example below.

What `imager` does not do:

`imager` does not handle calibration of visibility data beyond correction for parallactic angle variations. Instead, you should use the calibrater tool for this purpose. However, `imager` and calibrater can cooperate on the self-calibration of data.

What improvement to `imager` are in the works:

We are currently working on a number of improvements:

- Improved gridder to handle many telescopes and many channels more efficiently.
- Parallelized CLEAN and gridding

Advanced use of imager:

As with all CASA applications, `imager` is designed to be open: all the results are written to and read from standard CASA table files. This open design of `imager` also allows the user to try out new methods of processing data. Models may be read into Glish, edited or manipulated via standard Glish facilities, and then written out and used subsequently in `imager`. Suppose that we want to halve the Stokes I of all pixels with negative Stokes I. The following Glish fragment does the trick:

```
m:=image('myimage')
shape:=im.shape()
blc:=[1,1,1,1]
trc:=[shape[1],shape[2],1,shape[4]]
a:=m.getchunk(blc,trc)
a[a<0.]*:=0.5
m.putchunk(a,blc)
m.flush()
m.close()
```

Overview of `imager` tool functions:

Data access open, close, done

Data selection setdata

Data editing clipvis

Data calibration correct

Data examination plotvis, plotuv, plotweights, plotsummary

Weighting weight, filter, uvrange, sensitivity, fitpsf, plotweights

Image definition advise, setimage, make

Imaging makeimage, clean, nnls, mem, pixon, restore, residual, approximatepsf, fitpsf, setbeam, ft, smooth, feather, makemodelfromsd

Masks mask, boxmask, regionmask, exprmask, clipimage

Miscellaneous summary, setoptions, setoptions, setoptions

Example The following example shows the quickest way to make a CLEAN image and display it. Note that this can be more easily done from the toolmanager.

```
include 'imager.g'
#
# First make the MS from a FITS file:
#
m:=fitstoms(msfile='3C273XC1.MS', fitsfile='3C273XC1.FITS'); m.close();
#
# Now make an imager tool for the MS
#
imgr:=imager('3C273XC1.MS')
#
# Set the imager to produce images of cellsize 0.7 and
# 256 by 256 pixels
#
imgr.setimage(nx=256,ny=256, cellx='0.7arcsec',celly='0.7arcsec');
#
# Wait for results before proceeding to the next step
#
dowait:=T
#
# Make and display a clean image
#
imgr.clean(niter=1000, threshold='30mJy',
model='3C273XC1.clean.model', image='3C273XC1.clean.image')
dd.image('3C273XC1.clean.image')
#
# Fourier transform the model
#
imgr.ft(model='3C273XC1.clean.model')
#
# Plot the visibilities
#
imgr.plotvis()
#
# Write out the final MS and close the imager tool
```

```
#
imgr.close()
```

imager-Tool.html

2.3.1 imager - Tool

tool for synthesis imaging
Requires:

Synopsis

Description

imager is an tool that accomplishes synthesis processing. A `imager` must be constructed for each MeasurementSet for which one wishes to do processing. Multiple copies of `imager` may be made at any time (provide they are given different names).

Methods

imager	Construct an imager tool
advise	Advise (and optionally use) parameter values
advisechansel	Advise on spw and chan selection optimal for the image frequency range wanted
approximatepsf	Calculate approximate point spread functions
boxmask	Construct a mask image from blc, trc
calcuvw	Calculates (u, v, w) coordinates for the ms.
clean	Calculate a deconvolved image with selected clean algorithm
clipimage	Zero all pixels where Stokes I is below a threshold
clipvis	Flag visibilities where residual exceeds a threshold
close	Close the imager tool, with data written on disk, keeping imager process running for fu
defineimage	Set the image parameters for subsequent processing
done	Terminate the imager process
drawmask	Allows you do draw mask using the viewer
exprmask	Construct a mask image from a LEL expression
feather	Feather together an interferometer and a single dish image in the Fourier plane
filter	Apply additional weighting by filtering (u-v taper)
fitpsf	Fit the point spread function, making psf image first if needed
fixvis	Performs visibility adjustments.
ft	Fourier transform the specified model and componentlist
getweightgrid	get the requested weight grids
linearmosaic	Make a linear mosaic of several images
make	Make an empty (i.e. blank) image
predictcomp	Make a component list for a known object
makeimage	Calculate images by gridding, etc.

makemodelfromsd	Make an initial model image from a Single Dish image
mask	Construct a mask image by thresholding an image
mem	Calculate a deconvolved image with selected mem (maximum entropy) algorithm
nnls	Calculate a deconvolved image using the NNLS algorithm
open	Open a new MeasurementSet, for processing, closing current MeasurementSet
pb	Applies or corrects for a primary beam
plotsummary	Plot a summary of field and spectral window ids
plotuv	Plot the uv coverage
plotvis	Plot the visibility amplitudes as a function of u-v radius (also, see visplot tool
plotweights	Plot the visibility weights as a function of u-v radius
regionmask	Construct a mask image from a region
regiontoimagemask	union a mask image with various regions
residual	Calculate the residual image with respect to current model and component list
restore	Calculate the restored image with restored model, component list, and residuals
updateresidual	Calculate the residual and restored images with new modified model, component list,
sensitivity	Calculate rms sensitivity
apparentsens	Calculate rms sensitivity directly from weights
setbeam	Set the beam parameters for clean restoration
selectvis	Select visibilities for subsequent processing
setjy	Compute the model visibility for a specified source flux density
ssoflux	Use setjy instead.
setmfcontrol	Set various cycle control parameters for multi-field and wide-field imageing.
setoptions	Set some general options for subsequent processing
setscales	Set the scale sizes for MultiScale Clean
setsmallscalebias	Set bias toward smaller scales for MultiScale Clean
settaylorterms	Set the number of Taylor series terms for Multi-Frequency Clean
setsdoptions	Set some options for single dish processing
setvp	Set the voltage pattern model for subsequent processing
setweightgrid	set the requested weight grids
smooth	Calculate an image smoothed with a Gaussian beam
stop	stop the currently executing function asap
summary	Summarize the current state of the imager tool
uvrange	Select data within the limit of a given range
weight	Apply additional weighting to the visibility weights
mapextent	Compute map extent from given set of MSs

imager.imager.html

imager.imager - Function

2.3.1 Construct an imager tool

Description

This is used to construct `imager` tools associated with a MeasurementSet. The `imager` tool may then be used to generate various types of images. Note that a new executable is started every time the constructor is called.
This returns a Glish variable containing the tool functions of imager in an alternate universe that you have to tunnel to with a wormhole

Arguments

Inputs		
filename	MeasurementSet to be imaged	
	allowed:	string
	Default:	
compress	Compress calibration columns?	
	allowed:	bool
	Default:	Bool F
host	Host on which to run imager	
	allowed:	string
	Default:	
forcenewserver	Flag to force a new imager client	
	allowed:	bool
	Default:	true

Returns

imager

Example

```
im.open('3C273XC1.MS')
```

```
im.defineimage(nx=256, ny=256, cellx='0.7arcsec', celly='0.7arcsec')
im.image(type='corrected', image='3C273XC1.dirty')
im.close()
```

---

imager.advise.html

imager.advise - Function

2.3.1 Advise (and optionally use) parameter values

Description

Advise on recommended values of certain parameters. Return these values and optionally use them in Imager.
The calculations are performed as following:

cell The maximum uv distance in wavelength is found and then half of the inverse is taken as the maximum cellsize allowed.

pixels The field of view is converted to a number of pixels using the calculated cell size.

facets The number of facets on an axis is calculated in two different ways. The first method simply requires that the peeling of facets away from the celestial sphere should not cause an amplitude drop of more than the argument `amplitudeloss`. The positions may be incorrect, but all the sources will be removed correctly. The second method requires that the source positions be accurate to the same fraction of the beam specified by `amplitudeloss`. The second calculates the second moment in w and in uv distance and chooses the number of facets correspondingly. The first method does the same but after fitting a plane to the sampling: $w = au + bv$. For an approximately coplanar array, the positions may be wrong but the removal of sidelobes will be accurate. The number of facets returned is the second, usually smaller, number. The formula used is:

$$N\_facets = N\_pixels \sqrt{\frac{\frac{\Delta\theta}{\sqrt{8\delta A}} w\_rms}{uv\_rms}} \tag{2.2}$$

where $\Delta\theta$ is the cellsize in radians, and δA is the amplitude loss. This formula can be derived from (a) the peeling of facets from the celestial sphere, and (b) a quadratic approximation for the beam size both in the plane of the sky and along the w axis.

Arguments

Outputs		
pixels	Number of pixels on a side	
	allowed:	int
	Default:	
cell	Recommended maximum cellsize	
	allowed:	record
	Default:	
facets	Recommended number of facets on one axis	
	allowed:	int
	Default:	
phasecenter	Direction of phase center as a measure	
	allowed:	string
	Default:	
Inputs		
takeadvice	Use the advised values?	
	allowed:	bool
	Default:	true
amplitudeloss	Maximum fractional amplitude loss due to faceting	
	allowed:	double
	Default:	0.05
fieldofview	Desired field of view	
	allowed:	any
	Default:	variant 1.0deg

Returns
bool

---

imager.advisechansel.html

imager.advisechansel - Function

2.3.1 Advise on spw and chan selection optimal for the image frequency range wanted

Description

Basically tells you what channels of which spectral window need to be selected for your image spectral parameters. The freqstep is used to calulate the extra padding needed for data selection at the begining and end of the range. The freqframe parameter is the frame in which the frequency range is being given. It will be converted to the frame of the data with time to locate which channel match. A record will be returned with an element for each ms used in selectvis. Each element of the record will have the spwids and channel start and nchan for each spwid. if the parameter msname is used then the MSs associated associated with this tool (that have been either 'open'ed or 'selectvis'ed) are ignored In this mode it is a helper function to the general world ...no need to open or selectvis. You need to specify the field_id for which this calculation is being done for in the helper mode. If you have already set MS's and selected data and msname="" then the calulation is done for the field(s) selected in selectvis.
If the parameter `getfreqrange=True` then the reverse is requested. You set `spwselection` to be the range of data selection you want to use and you'll get the range of frequency covered in the frame you set.

Arguments

Inputs		
freqstart	Begining of frequency range in Hz	
	allowed:	double
	Default:	1.0e6
freqend	End of frequency range in Hz	
	allowed:	double
	Default:	1.1e6
freqstep	spectral channel resolution of intended image in Hz	
	allowed:	double
	Default:	100.0
freqframe	frame in which frequency is being expressed in other parameters	
	allowed:	string
	Default:	LSRK
msname	name of an ms, if empty string it will use the ms's used in selectvis	
	allowed:	string
	Default:	
fieldid	fieldid to use when msname is not empty otherwise ignored and field selected in selectvis is used	
	allowed:	int
	Default:	0
getfreqrange	if set then freqrange is returned in the frame requested for the data selected	
	allowed:	bool
	Default:	false
spwselection	if getfreqrange=True then this is needed to find the range of frequency in the frame requested	
	allowed:	string
	Default:	

Returns
record

Example

```
In this example, we are interested in an image cube which span 20.0682GHz to 20.1982 in LSRK
```

```
#######
im.selectvis(vis='test1.ms', field='4', spw='*')
```

```
im.selectvis(vis='test2.ms', field='4', spw='*')
selinfo=im.advisechansel(freqstart=2.00682e10, freqend=2.01982e10, freqstep=3.9e3, freqframe
####The output "selfinfo" will be a record which will look like thus
{'ms_0': {'nchan': array([109,  23], dtype=int32),
          'spw': array([4, 5], dtype=int32),
          'start': array([19,  0], dtype=int32)},
 'ms_1': {'nchan': array([109,  23], dtype=int32),
          'spw': array([4, 5], dtype=int32),
          'start': array([19,  0], dtype=int32)}}
###
Thus from the first ms a spw selection like  '4:19~127, 5:0~22'  is all that is needed.
Similarly from the second ms.

###if you need this info without needing to change the state of the imager tool
then you can it as follows

im.advisechansel(freqstart=2.00682e10, freqend=2.01982e10, freqstep=3.9e3, freqframe="LSRK"


####now if you want to see what frequency range is covered, in the frame
####defined by freqframe, in spwselection you want to use
im.selectvis(vis='test3.ms', spw='0:20~210')
im.advisechansel(getfreqrange=True, freqframe="LSRK")
### the output will be something
 {'freqend': 346020345384.64178, 'freqstart': 345683852920.1723}

###and if you just want to use it as a helper function without touching the state
###of imager

im.advisechansel(msname='test3.ms', getfreqrange=True, spwselection='0:20~210')
```

---

imager.approximatepsf.html

imager.approximatepsf - Function

2.3.1 Calculate approximate point spread functions

Description

Calculate the approximate point spread function. *Note that the model visibilities are updated.*
Some types of imaging do not yield a well-defined point spread function. For example, mosaicing or single dish imaging both yield point spread functions that are position dependent. Nevertheless, one can still usefully define an *approximate* PSF that is of some utility. This is calculated by doing the following calculation: a point source is located at the center of the specified coordinate system and the model data predicted. The approximate PSF is then formed from those model data using the full sky equation. For regular sampling in the image plane, this approximate PSF is actually quite good. It can be used in a deconvolution. For a mosaic with similar uv sampling per pointing, the approximate PSF is roughly the PSF per pointing multiplied by the primary beam. For a single dish image, it is roughly the telescope primary beam convolved with itself (if the gridfunction='pb' was selected).

Arguments

Inputs		
psf	Name of output point spread function	
	allowed:	string
	Default:	
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns

bool

Example

Example of how to make the approximate psf for a mosaic:

```
im.open('orion.ms')
im.selectvis(spwid=[0, 1] ,field=range(2,11));
im.defineimage(nx=300, ny=300, cellx='2.0arcsec',celly='2.0arcsec' , stokes="I", phasecente
im.weight('natural')
im.setvp(dovp=T, usedefaultvp=True)
im.setoptions(ftmachine='mosaic', padding=1.0)
im.approximatepsf(psf='LePSF.image')
```

---

imager.boxmask.html

imager.boxmask - Function

2.3.1 Construct a mask image from blc, trc

Description

A mask image is an image with the same shape as the other images but with values between 0.0 and 1.0 as a pixel value. Mask images are used in imager to control the region selected in a deconvolution.
In the Clark CLEAN, the mask image can usefully have any value between 0.0 and 1.0. Intermediate value discourage but do not rule out selection of clean components in that region. This is accomplished by multiplying the residual image by the mask prior to entering the minor cycle. Note that if you do use a mask for the Clark or Hogbom Clean, it must cover only a quarter of the image. boxmask does not enforce this requirement.

Arguments

Inputs		
mask	name of mask image	
	allowed:	string
	Default:	
blc	Bottom left corner	
	allowed:	intArray
	Default:	0 0 0 0
trc	Top right corner, should be image shape	
	allowed:	intArray
	Default:	
value	Value to fill in	
	allowed:	double
	Default:	1.0

Returns
bool

Example

```
im.boxmask(mask='bigmask', blc=[56,45,1,1], trc=[87,93,4,1])
im.clean(mask='bigmask', model='3C273XC1.clean.masked', niter=1000)
```

Makes the image bigmask, and then sets it to unity for all points in the region bounded by the blc and trc. Then cleans using it as the mask.

---

imager.calcuvw.html

imager.calcuvw - Function

2.3.1 Calculates (u, v, w) coordinates for the ms.

Description

This calculates (u, v, w) positions for the visibilities using the antenna and feed positions and offsets, the time, and the phase tracking center(s).

Arguments

Inputs	
fields	Field IDs (numbered relative to 0) to operate on. Blank = all. allowed: intArray Default: -1
refcode	Reference frame to use for the generated (u, v, w)s. WARNING: clean and the im tool ignore the reference frame claimed by the UVW column (it is often mislabelled as ITRF when it is really J2000) and instead assume the (u, v, w)s are in the same frame as the phase tracking center. calcuvw does not yet force the UVW column and field centers to use the same reference frame! Blank = use the phase tracking frame of vis. allowed: string Default:
reuse	Start from the UVWs in vis (True) or calculate them from the antenna positions? allowed: bool Default: true

Returns

bool

Example

```
im.open("3C273XC1.MS")
im.calcuvw()
im.done()
```

---

imager.clean.html

imager.clean - Function

2.3.1 Calculate a deconvolved image with selected clean algorithm

Description

Makes a clean image using either the Hogbom, Clark, multi-scale or multi-field algorithms. The Clark algorithm is the default. The clean is performed on the residual image calculated from the visibility data currently selected. Hence the first step performed in clean is to transform the current model or models (optionally including a componentlist) to fill in the MODEL_DATA column, and then inverse transform the residual visibilities to get a residual image. This residual image is then cleaned using the corresponding point spread function. This means that the initial model is used as the starting point for the deconvolution. Thus if you want to restart a clean, simply set the model to the model that was previously produced by clean.
Rather than explicit CLEAN boxes, mask images are used to constrain the region that is to be deconvolved. To make mask images, use either boxmask (to define a mask via the corner locations blc and trc) or mask (to define a mask via thresholding an existing image) or regionmask (to make masks via regions using the regionmanager or interactively through the viewer) . The default mask is the inner quarter of the image.
The CLEAN deconvolution is joint in whatever Stokes parameters are present. Thus it searchs for peaks in either I or $I + |V|$ or $I + \sqrt{Q^2 + U^2 + V^2}$, the rationale for the latter two forms being to be biased towards finding strongly polarized pixels first (these forms are also the maximum eigenvalue of the coherency matrix). The PSF is constrained to be the same in all polarizations (a feature of this implementation, not of the Hamaker-Bregman-Sault formalism). But the option of searching peaks in the stokes planes independently is available via the `clarkstokes` parameter
The clean algorithms possible are:

Hogbom The classic algorithm: points are found iteratively by searching for the peak. Each point is subtracted from the full residual image using the shifted and scaled point spread function.

Multiscale An experimental multi-scale clean algorithm is invoked. The algorithm is fully described in deconvolver.

Clark The faster algorithm: the cleaning is split into minor and major cycles. In the minor cycles only the brightest points are cleaned, using a subset of the point spread function. In the major cycle, the points thus found are subtracted correctly by using an FFT-based convolution.

Multi-field Cleaning is split into minor and major cycles. For each field, a Clark-style minor cycle is performed. In the major cycle, the points thus found are subtracted either from the original visibilities (for multiple fields) or using a convolution (for only one field). The latter is much faster. Multi-field imaging has been implemented for Clark, Hogbom, and Multi-scale deconvolution algorithms.

Cotton-Schwab Cleaning is split into minor and major cycles. For each field, a Clark-style minor cycle is performed. In the major cycle, the points thus found are subtracted from the original visibilities. A fast variant does a convolution using a FFT. This will be faster for large numbers of visibilities. Double the image size from that used for Cotton-Schwab and set a mask to clean only the inner quarter.

Wide-field The user will need to use a wide-field algorithm to deconvolve if the array is not coplanar over the field of view being imaged . The technique used is to break the field being imaged into smaller pieces (facets), over each of which the array appear planar. We implement a rectangular facetting scheme. If the number of facets specified in defineimage is greater than one, Either wfhogbom or wfclark algorithm has to be selected here to perform a wide-field decovolution. The function advise can be used to calculate or check if you need to use a wide-field deconvolution. Note that aliasing can be reduced by using the `padding` argument in setoptions. In practice the previous sentence means that if you notice the clean to diverge at the edges of the facets then you need to use a larger amount of padding for the FT; the default being 1.2. Wide-field imaging has been implemented for Clark and Hogbom algorithms.

The multi-field clean should be used if either of two conditions hold:

1. Multiple fields are to be cleaned simultaneously **OR**
2. Primary beam correction is enabled. In this case, a mosaiced clean is performed.

Note that for the single pointing algorithms, only a quarter of the image may be cleaned. If no mask is set, then the cleaned region defaults to the inner quarter. If a mask larger than a quarter of the image is set, then only the inner quarter part of that mask is used. However, for the wide-field and multi-field imaging (including the Cotton-Schwab algorithm), the entire field may be imaged because the major cycles either do an exact subtraction from the visibilities or because PSF extent is more than twice the extent of the primary beam support.
Before `clean` can be run, you must run `selectvis` and `defineimage`. Before `clean` can be run with a multi-field algorithm (especially for mosaic), you should run `setvp`. You may want to run `setmfcontrol` before running `clean` with a multi-field or wide-field algorithm, though the default control values

may be acceptable. Before `clean` can be run with a multi-scale algorithm, `setscales` must be run.
Interactive cleaning/masking: If the user wants to see what the clean image looks like after npercycle iteration and mask or modify the mask each time, he/she should set `interactive=True` and give npercycle to a fraction of niter. A viewer with the last residual image along with an overlayed mask appear after every npercycle iteration. The user can add or delete regions (by clicking on the appropriate button) to the mask using the region button and drawing regions and double clicking inside the region. When satisfied and ready to continue cleaning press 'DONE with masking' (if the user want to terminate the cleaning process use the 'STOP' button). The button 'No more mask changes' should be used if the user want clean to proceed without any further interruption. Even if `interactive=False`, and if the parameter 'mask' is non-empty, it is still used in limiting the search area for clean components. If the parameter 'masktemplate' is not empty this means that the user want to use an apriori image to make the mask the first time (e.g a previously cleaned image)
This function returns a record containing convergence, iterations used and threshold reached.

Arguments

Inputs		
algorithm	Algorithm to use	
	allowed:	string
	Default:	clark
		clarkstokes
		hogbom
		multiscale
		mfclark
		mfclarkstokes
		csclean
		csfast
		mfhogbom
		mfmultiscale
		wfclark
		wfhogbom
		clark
niter	Number of Iterations, set to zero for no CLEANing	
	allowed:	int
	Default:	1000
gain	Loop Gain for CLEANing	
	allowed:	double
	Default:	0.1
threshold	Flux level at which to stop CLEANing	
	allowed:	any
	Default:	variant 0.0Jy
displayprogress	Display the progress of the cleaning?	
	allowed:	bool
	Default:	false
model	Names of clean model images	
	allowed:	stringArray
	Default:	
keepfixed	Keep one or more models fixed	
	allowed:	boolArray
	Default:	false
complist	Name of component list	
	allowed:	string
	Default:	
mask	Names of mask images used for CLEANing	
	allowed:	stringArray
	Default:	
image	Names of restored images	
	allowed:	stringArray
	Default:	
residual	Names of residual images	
	allowed:	stringArray
	Default:	
psfimage	Names of psfs if they are needed	
	allowed:	stringArray
	Default:	
interactive	whether to stop clean and interactively mask	
	allowed:	bool
	Default:	false
npercycle	If interactive is 'T', then no of iter of clean before stopping, usually a fraction of niter	
	allowed:	int
	Default:	100

Returns
record

Example

```
im.clean(model='3C273XC1.clean.model',
mask='3C283XC1.mask', niter=1000, gain=0.25, threshold='0.03Jy')


A few points should be noted in this example:

\begin{itemize}
\item When the mask parameter is specified, the number of mask images
      listed should be equal to the number of model images.  They
      should also have the same coordinate system as their
      corresponding model images.
\item If one or more model images are listed in the model parameter
      but the image and residual parameters are empty, the restored
      and residual images are automatically named as the model names
      appended with '.restored' and '.residual', respectively.
\item No restored or residual image is made if the respective image
      string is explicitly unset.
\end{itemize}


include 'imager.g';
msfile = 'vlac125K.ms';
im.open(msfile);
npix = 500; cell='5arcsec';
#
# CS on 500 by 500
#
im.defineimage(nx=npix, ny=npix, cellx=cell, celly=cell, stokes='I',
      spw=[0,1]);
im.setoptions(padding=1.0);
im.selectvis(spwid=[0,1]);
im.clean('cs', model='vlac125K.cs', image='vlac125K.cs.restored',
   niter=1000, gain=0.1);
#
# CSF on 1000 by 1000, cleaning a given box
#
```

```
im.defineimage(nx=2*npix, ny=2*npix, cellx=cell, celly=cell, stokes='I',
      spwid=[1,2]);
reg=rg.box(blc=[400,500], trc=[450,550])
im.regionmask('vlac125K.mask', region=reg);
im.clean('csf', model='vlac125K.csf', image='vlac125K.csf.restored',
   mask='vlac125K.mask', niter=1000, gain=0.1);

#
# CS on 1000 by 1000, cleaning entire image
#

im.defineimage(nx=2*npix, ny=2*npix, cellx=cell, celly=cell, stokes='I',
      spwid=[1,2]);
im.clean('cs', model='vlac125K.csl', image='vlac125K.csl.restored',
   mask='vlac125K.mask', niter=1000, gain=0.1);

im.done();
```

---

imager.clipimage.html

imager.clipimage - Function

2.3.1 Zero all pixels where Stokes I is below a threshold

Description

All pixels in the image with Stokes I less than some threshold are set to zero. This is useful prior to self-calibration where one oftens wishes to remove negative pixels from the model. Note that if the image has polarization information, then the polarized part of a pixel is also set to zero if Stokes I is less than the threshold.

Arguments

Inputs		
image	name of image	
	allowed:	string
	Default:	
threshold	Threshold	
	allowed:	any
	Default:	variant 0.0Jy

Returns

bool

Example

```
im.clipimage(image='clean', threshold='50mJy')
```

---

imager.clipvis.html

imager.clipvis - Function

2.3.1 Flag visibilities where residual exceeds a threshold

Description

All visibilities where the residual exceeds some threshold are flagged. This provides a simple way of flagging bad data.

Arguments

Inputs		
threshold	Threshold	
	allowed:	any
	Default:	variant 0.0Jy

Returns

bool

Example

```
im.plotvis('residual')
# determine threshold then apply it
im.clipvis(threshold='50mJy')
```

---

imager.close.html

imager.close - Function

2.3.1 Close the imager tool, with data written on disk, keeping imager process running for future use

Description

This is used to close `imager` tools. Note that the data is written to disk. The `imager` process keeps running until a done tool function call is performed.

Arguments

Returns
bool

Example

```
im.open('3C273XC1.MS')
im.makeimage(image='3C273XC1.dirty',type='corrected')
im.close()
```

---

imager.defineimage.html

imager.defineimage - Function

2.3.1 Set the image parameters for subsequent processing

Description

Define the default image parameters. If an image is to be made, then these parameters are used in the construction of the image. Thus, for example, the tool function make makes an (empty) image using these parameters.
Note that some parameters can be specified either in canonical units or via measures. To establish default values, the ids for the default spectral window and default field id must be given.
The parameter `mode` can be one of the following:

- mfs
- channel
- velocity or opticalvelocity
- frequency

`imager` can perform multi-frequency synthesis over several spectral windows (mode='mfs'). To achieve this, you should set spwid to an array of the required spectral windows (*e.g.* `spwid=[0,1]`).
WARNING: For multifrequency synthesis, 'mfs', it is important that the spwid's selected in selectvis be the SAME as the one selected in `defineimage`. Otherwise the frequency at which the image is made is not going to be the same as to the one as the one used in gridding the visibility and can lead to image artifacts. For `mode='velocity'` and `mode='frequency'` the `step` parameter has to be a measure/quantity of velocity or frequency, otherwise for `mode='channel'` step is the number of data channels to be averaged to make one image channel(see examples below).
The phase center of the image defaults to that of the specified phasecenter (the first fieldid in the ms is taken if none is specified), this parameter can be a fieldid or a measure string or the record output from the direction function of the measures tool(direction). This is important if you have multiple pointings in the data. The user would have used selectvis to select which pointings would be used in imaging. If the conversion from the observed direction requires frame information then this is taken as follows:

- Direction information, including the coordinate system, is taken from the relevant entry in the Field table of the MeasurementSet.

- The epoch is taken from the time of observation of each visibility.
- A position is specified via the `imager` tool function setoptions

If the specified number of facets is greater than unity then the image is split into facets (this number along the x and y axes) and processed. This is necessary when using wide-field algorithm for deconvolving the image, in cases of non-coplanar arrays (e.g the VLA at low frequencies but can be safely left at 1 for the ATCA or WSRT). This is now recommended only when memory or image size is of a problem, otherwise for widefield issues, wprojection (ftmachine parameter in setoptions) is recommended with a single facet.
For spectral imaging `defineimage` and `selectvis` defines the spectral channels that are imaged. Examples are given in the selectvis section. The parameter `restfreq` can be used to define what rest frequency to use in the resulting images. If none is specified imager will try to use the one that is defined in the ms. It will use the first one defined in the first spectral window selected.
For wide-field or 3D imaging see setoptions section for some examples.
If the telescope is observing moving source (e.g planet or moon) over a period of time. One may wish to image in a frame where the source is fixed. The parameter `movingsource` is for that. Setting it to a source that `measures` is aware of will force the imaging to realign (shift in SD imaging or phase rotation in interferometry imaging) the data so that the source appears fixed in the image. Obviously in doing so the background sources will be blurred. The coordinate system used to fix the source on is the one where the source is at the first time observed in the selected data.

Arguments

Inputs		
nx	Total number of spatial pixels in x	
	allowed:	int
	Default:	128
ny	Total number of spatial pixels in y	
	allowed:	int
	Default:	-1
cellx	Cellsize in x (e.g. '1arcsec')	
	allowed:	any
	Default:	variant 1.0
celly	Cellsize in y (e.g. '1arcsec')	
	allowed:	any
	Default:	variant
stokes	Stokes parameters to image (e.g. 'IQUV')	
	allowed:	string
	Default:	IV
		IQU
		IQUV
		I
phasecenter	Direction of phase center as a diretion measure or a field id	
	allowed:	any
	Default:	variant 0
mode	Type of processing (velocity =radiovelocity)	
	allowed:	string
	Default:	frequency
		radiovelocity
		opticalvelocity
		truevelocity
		mfs
nchan	Number of channels; a -1 (default) means all the channels as selected in selectvis and combined into one continuum channel	
	allowed:	int
	Default:	-1
start	Start channel; A 0-relative channel number of the spwid or a frequency quantity or a velocity quantity or radial velocity measure	
	allowed:	any
	Default:	variant 0
step	Step in channel; integer for number of channels or frequency quantity or velocity quantity or radial velocity measure	
	allowed:	any
	Default:	variant 1
spw	Spectral Window Id (0 relative) that defines center of image	
	allowed:	intArray
	Default:	0
restfreq	rest frequency to use; default =& use the one available in ms	
	allowed:	any
	Default:	variant
outframe	frequency frame of output image (default LSRK, "" =>	

Returns
bool

Example

```
## Example 1
im.defineimage(nx=1024,ny=1024, cellx='30marcsec',celly='30marcsec',
nchan=1, stokes='IV', phasecenter=me.direction('mars'));
## Example 2
im.defineimage(nx=1024,ny=1024, cellx='30marcsec',celly='30marcsec',
nchan=1, stokes='IV', phasecenter=['J2000', '19:00:30.5', '-45d00m25.6']);
## Example 3
im.selectvis(nchan=10, start=3, spw=[0,1], field=[3, 4, 5, 6, 7, 9, 10])
im.defineimage(nx=500, ny=500, mode='mfs', spwid=[0,1], fieldid=7)
im.clean(algorithm='mfclark', niter=1000, model='mosaic.model', image='mosaic.image')

## Example 4

dir1=me.direction('J2000', '20h00m00', '21d00m00')
dir2=me.direction('J2000', '20h10m00', '21d00m00')
dir3=me.direction('J2000', '20h00m00', '21d03m00')
im.defineimage(nx=100,  cellx='0.1arcsec', phasecenter=dir1)
im.make('box1')
im.defineimage(nx=100, cellx='0.1arcsec', phasecenter=dir2)
im.make('box2')
im.defineimage(nx=100, cellx='0.1arcsec', celly='0.1arsec', phasecenter=dir3)
im.make('box3')
im.clean(algorithm='mfclark', model=['box1', 'box2', 'box3'],
   image=['box1.restored', 'box2.restored', 'box3.restored'],
   residual=['box1.residual', 'box2.residual', 'box3.residual'])
```

In the first example, the image parameters are set for 1024 by 1024 pixels of 30milli arcsec, 1 channel will be made, Stokes I and V will be imaged, and the phasecenter will be the direction of Mars as given by the JPL DE-200 emphemeris. In the second, the phase center is taken to be an absolute coordinate value.

The third example shows the use of selectvis and defineimage to setup a mosaic. In the set data we have chosen 10 channels (for each spectral window) of data starting form

channel 3. We also have selected spectral windows 0 and 1. We have selected data from fields 3 to 10. In the defineimage we decide to use the data to make a multifrequency synthesis image. We center the image on the field 7 pointing.

The fourth example is use to clean regions where the user knows the sources are and ignore all the other regions. This is very efficient in large fields with few sources. Smaller outlier images are made and deconvolved around known sources rather than making a big image englobing all three fields.

Now here are some examples about defining cubes using different {\tt mode} parameters.

defining channels cubes use the channel as defined in the data

```
im.defineimage(cellx=1000, mode='channel', nchan=100, start=10,
step=1, spwid=range(0,10))
```

now using frequency and overiding the rest frequency defined in the ms or if its not defined in the ms

```
im.defineimage(cellx=1000, mode='frequency', nchan=100, start='1GHz',
step='10kHz', restfrequency='1.421GHz')
```

in case you have a frame with the frequency

```
im.defineimage(cellx=1000, mode='frequency', nchan=100, start=['LSRK',
'1GHz'], step='10kHz', restfrequency='1.421GHz')
```

OR using measures

```
freqstart=me.frequency('LSRK', '1GHz')
im.defineimage(cellx=1000, mode='frequency', nchan=100,
start=freqstart, step='10kHz', restfrequency='1.421GHz')
```

similarly if you want to use velocity to define your cube

```
im.defineimage(cellx=1000, mode='velocity', nchan=100, start=['LSRK',
'10km/s'], step='1m/s', restfrequency='1.421GHz')
```

OR using measures

```
velstart=me.radialvelocity('LSRK', '10km/s')
im.defineimage(cellx=1000, mode='velocity', nchan=100, start=velstart,
step='1m/s')
```

Change mode to 'opticalvelocity' if your velocity values are using optical definition

---

imager.done.html

imager.done - Function

2.3.1 Terminate the imager process

Description

This is used to totally stop the `imager` process. It is a good idea to conserve memory use on your machine by stopping the process once you no longer need it.

Arguments

Returns
bool

Example

```
im.open('3C273XC1.MS')
im.makeimage(image='3C273XC1.dirty',type='corrected')
im.done()
```

---

imager.drawmask.html

imager.drawmask - Function

2.3.1 Allows you do draw mask using the viewer

Description

A mask image is an image with the same shape as the other images but with values between 0.0 and 1.0 as a pixel value. Mask images are used in imager to control the region selected in a deconvolution.
drawmask is used to interactively draw regions over a template image which you want to allow deconvolution to occur.

Arguments

Inputs		
image	name of template image	
	allowed:	string
	Default:	
mask	name of image to save mask in	
	allowed:	string
	Default:	
niter	Total number of iteration to display in box; just for display or python packaging	
	allowed:	int
	Default:	0
npercycle	npercycle value to display in box; just for display or python packaging	
	allowed:	int
	Default:	0
threshold	threshold to display in box ; just for display or python packaging	
	allowed:	string
	Default:	0 mJy

Returns

record

Example

```
im.drawmask(image='mytemplate.image', mask='myregions.mask')
im.clean(mask='myregions.mask', model='3C273XC1.clean.masked', niter=1000)
```

Make mask image by drawing interactively over a given image 'mytemplate.image', then image "clean regions".

---

imager.exprmask.html

imager.exprmask - Function

2.3.1 Construct a mask image from a LEL expression

Description

A mask image is an image with the same shape as the other images but with values between 0.0 and 1.0 as a pixel value. Mask images are used in imager to control the region selected in a deconvolution.
In the Clark CLEAN, the mask image can usefully have any value between 0.0 and 1.0. Intermediate value discourage but do not rule out selection of clean components in that region. This is accomplished by multiplying the residual image by the mask prior to entering the minor cycle. Note that if you do use a mask for the Clark or Hogbom Clean, it must cover only a quarter of the image. boxmask does not enforce this requirement.
This function allows Lattice Express Language (LEL) expressions to be used in defining a mask. See the documentation on imagecalc for more details.

Arguments

Inputs		
mask	name of mask image	
	allowed:	string
	Default:	
expr	Value to set the mask to. Any scalar or LEL expression	
	allowed:	double
	Default:	1.0

Returns
bool

Example

```
im.exprmask(mask='bigmask', expr='3C273XC1.clean>0.5')
im.clean(mask='bigmask', model='3C273XC1.clean.masked', niter=1000)
```

```
Makes the image bigmask, and then sets it to unity for all points in
the region where 3C273XC1.clean is greater than 0.5Jy.
Then cleans using it as the mask.
```

---

imager.feather.html

imager.feather - Function

2.3.1 Feather together an interferometer and a single dish image in the Fourier plane

Description

Basically the "imerg" algorithm of AIPS and SDE, or the "feather" algorithm of MIRIAD, we regrid the total power (or low resolution) image onto the interferometer (or high resolution) image, Fourier transform both the interferometer and single dish images, down weight the Fourier transform of the interferometer image by 1.0 - FT(low res psf), add the weighted interferometer Fourier plane to the single dish Fourier plane, and transform back into the image plane.
The tapering is by the transform of a point spread function. If lowpsf is specified, that image is used, otherwise the appropriate telescope beam is used. The point spread function for a single dish image may be calculated using makeimage.
`Advice:` Note that if you are feathering large images, you'd be advised to have the number of pixels along the X and Y axes to be composite numbers and definitely not prime numbers. In general FFTs work much faster on even and composite numbers. You may use subimage function of image tool to trim the number of pixels to something desirable.

Arguments

Inputs		
image	Name of output feathered image	
	allowed:	string
	Default:	
highres	Name of high resolution (interferometer) image	
	allowed:	string
	Default:	
lowres	Name of low resolution (single dish) image	
	allowed:	string
	Default:	
lowpsf	Name of optional low resolution point spread function	
	allowed:	string
	Default:	
effdishdiam	Optional new SD dish diameter in m to use in feathering; can be smaller than true dish size	
	allowed:	double
	Default:	-1.0
lowpassfiltersd	Reject the high spatial frequency of the SD image	
	allowed:	bool
	Default:	false
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
im.setvp(dovp=True, usedefaultvp=True)
im.feather(image='feathered.image', highres='casa.vlaonly',
lowres='casa.sd');

In the above example its using the default beams and the observatory
information is in the image header.

But if you have a single dish image with a beam which is not defined
in the casa database then the example below is a guide of how to do
that, say you know the beam of the single dish as a gaussian.
```

```
#create a beam pattern table using vpmanager
include 'vpmanager.g'
vpman=vpmanager();
vpman.setpbgauss(telescope='OTHER', othertelescope='BONN',
halfwidth='1arcmin', maxrad='20arcmin', reffreq='1.4GHz');
vpman.saveastable('bonn.pb')
vpman.done()

##....would have done  your usual imager setup (defineimage etc) then before feathering
im.setvp(dovp=True, usedefaultvp=false, vptable='bonn.pb')
im.feather(image='feathered.image', highres='casa.vlaonly',
lowres='casa.sd');

###
```

---

imager.filter.html

imager.filter - Function

2.3.1 Apply additional weighting by filtering (u-v taper)

Description

Apply visibility tapering to emphasize certain scale structures. The imaging tapers are applied to a Table column called IMAGING_WEIGHT, which may be plotted using tb and pl. plotweights. In addition, this column may be accessed directly using either the table or ms modules. Note that the taper is multiplicative and so the weights must be calculated first using weight. The points are not flagged!
Note that the scale size to be emphasized is given in the image plane as the parameters of the corresponding Gaussian. Note also use of this function provides an optimum detection for the given scale size, which is not the same as requiring that the resulting dirty beam have the specified Gaussian fit. The resultant fitted beam size will *very roughly* be the quadratic sum of the original beam and the specified beam. If you wish to obtain a specified beam, then the best approach is to perform this calculation and check the value obtained using imager.fitpsf.

Arguments

Inputs		
type	Type of filtering or u-v tapering	
	allowed:	string
	Default:	gaussian
bmaj	Major axis of filter	
	allowed:	any
	Default:	variant 1arcsec
bmin	Minor axis of filter	
	allowed:	any
	Default:	variant 1arcsec
bpa	Position angle of filter	
	allowed:	any
	Default:	variant 0deg
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
im.weight('uniform')
im.filter(type='gaussian', bmaj='2.3arcsec', bmin='1.67arcsec',
bpa='-34.5deg')
```

---

imager.fitpsf.html

imager.fitpsf - Function

2.3.1 Fit the point spread function, making psf image first if needed

Description

This fits an elliptical Gaussian to the point spread function and returns the fitted beam parameters. If psf image is not specified then a psf is made and used. The values for the beam fit are saved internally and used whenever needed (for example in the functions restore or smooth) until invalidated. The values are invalidated by selectvis, defineimage or any tool function that changes the weights. Use the function summary to check if there is a valid fitted psf stored internally.

Arguments

Outputs		
bmaj	Major axis of beam	
	allowed:	record
	Default:	
bmin	Minor axis of beam	
	allowed:	record
	Default:	
bpa	Position angle of beam	
	allowed:	record
	Default:	
Inputs		
psf	Name of input psf	
	allowed:	string
	Default:	
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
im.makeimage(type='psf', image='3C273XC1.psf')
params=im.fitpsf('3C273XC1.psf')
#This returns a python dict params here
print params['bmaj'].value, params['bmin'].value, params['bpa']
im.restore(model='bla' , complist='', image='bla.restored' , residual='bla2.residual' )
```

Or if one wants to generate a psf from the uv coverage and use that subsequently as in the

```
- im.fitpsf(psf='')
- im.restore(model='bla' , complist='', image='bla.restored' , residual='bla2.residual' )
```

---

imager.fixvis.html

imager.fixvis - Function

2.3.1 Performs visibility adjustments.

Description

Corrects UVW coordinates and optionally the visibilities for various effects that can be calculated without fitting a model to the data.
The effects include:

- changing the phase tracking center(s),
- correcting for differential aberration, (Not yet implemented)
- changing the equinox (i.e. B1950_VLA to J2000 or APP, etc.) of the UVW coordinates,
- changing the projection, as in (-)NCP to SIN. (Not yet implemented),
- refocusing.

Arguments

Inputs	
fields	Field IDs (numbered relative to 0) to operate on. Blank = all. allowed: intArray Default: -1
phasedirs	Phase tracking centers for each field in fields, in the same order. allowed: stringArray Default:
refcode	Reference frame to use for the generated UVWs. WARNING: clean and the im tool ignore the reference frame claimed by the UVW column (it is often mislabelled as ITRF when it is really J2000) and instead assume the (u, v, w)s are in the same frame as the phase tracking center. calcuvw does not yet force the UVW column and field centers to use the same reference frame! Blank = use the phase tracking frame of vis. allowed: string Default:
distances	A list of distances (in m) for the fields listed in fields. 0 = infinity. allowed: doubleArray Default: 0.0
datacolumn	Which of DATA, MODEL_DATA, and/or CORRECTED_DATA to operate on. Default: "all". allowed: string Default: all

Returns
bool

Example

```
im.open("3C273XC1.MS")
im.fixvis()
im.done()
```

imager.ft.html

imager.ft - Function

2.3.1 Fourier transform the specified model and componentlist

Description

Fourier transform the specified model (and optionally componentlist) and insert into the MODEL_DATA column. The current contents of the MODEL_DATA column are replaced unless incremental is set to T (in which case the results are added to the column).

Arguments

Inputs		
model	Name of image	
	allowed:	stringArray
	Default:	
complist	Name of component list	
	allowed:	string
	Default:	
incremental	Add to the existing MODEL_DATA column?	
	allowed:	bool
	Default:	false
async	Run asynchronously in the background?	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
im.ft(model='3C273XC1.nnls.model')
im.ft(model='3C273XC1.another.model', incremental=True)
```

```
Fourier transforms the model in the image 3C273XC1.nnls.model
and then adds the visibility due to 3C273XC1.another.model
```

---

imager.getweightgrid.html

imager.getweightgrid - Function

2.3.1 get the requested weight grids

Description

This is a utility function when running multi imager processes in parallel on subsection of an ms/data independently One would wish to weight the dirty image before averaging or set the imaging weight density (when using unform or Brigg's style weighting) to account for all the data being used. This is **NOT** for the general user but for people who are parallelizing at the scripting level.
imaging: will return a the weight griddensity
ftweight: will put the FT-machine weight images in the names given in wgtimage parameters..these may be needed to average residual images from different processes running seperately on different section of the data.

Arguments

Inputs		
type	Type of weight requested (imaging, ftweight)	
	allowed:	string
	Default:	imaging
wgtimages	names of weightimages to save	
	allowed:	stringArray
	Default:	

Returns
anyvariant

Example

```
wght=im.getweightgrid('imaging')
wght2=im2.getweightgrid('imaging')
wght=wght+wght2
```

```
im.setweightgrid(weight=wght, type='imaging')
```

---

imager.linearmosaic.html

imager.linearmosaic - Function

2.3.1 Make a linear mosaic of several images

Description

Make a linear mosaic of several images. Currently, the pointing center is not specified in the image, so we specify the pointing center in terms of the row numbers of the FIELD subtable.

Arguments

Inputs	
images	Input images to be mosaiced
	allowed: stringArray
	Default:
mosaic	Output mosaic image
	allowed: string
	Default:
fluxscale	Fluxscale image
	allowed: string
	Default:
sensitivity	Sensitivity image
	allowed: string
	Default:
fieldids	List of field ids that correspond each of the images,used to center the PB of each image. (0-based list)
	allowed: intArray
	Default: 0
usedefaultvp	Use the default vp type?
	allowed: bool
	Default: true
vptable	Voltage pattern table from the vpmanager for detailed specification
	allowed: string
	Default:
async	Run asynchronously in the background
	allowed: bool
	Default: false

Returns
bool

Example

```
im.linearmosaic(images=['orion.1.cln', 'orion.2.cln', 'orion.4.cln'], mosaic='orion.linmos',
fluxscale='orion.linmos.fluxscale', fieldid=[1,2,4]);
```

---

imager.make.html

imager.make - Function

2.3.1 Make an empty (i.e. blank) image

Description

Make an empty image using the current image parameters. Often this is unnecessary, but you will typically need to use this if you wish to deconvolve a set of images. The steps are to make the empty images that you require to be deconvolved, and then pass them into clean as a vector of strings.

Arguments

Inputs		
image	name of output image	
	allowed:	string
	Default:	
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
im.defineimage(nx=1024,ny=1024, cellx='30marcsec',celly='30marcsec',
nchan=1, stokes='IV', phasecenter=me.direction('mars'));
im.make('mars.moving');
im.defineimage(nx=1024,ny=1024, cellx='30marcsec',celly='30marcsec',
nchan=1, stokes='IV',
phasecenter=me.direction('J2000', '12:23:48.7', '-15:56:32.9')
im.make('mars.fixed');
im.clean(algorithm='mf', model=['mars.moving', 'mars.fixed'],
image=['mars.moving.restored', 'mars.fixed.restored'])
```

This makes two empty images, one moving with mars and one fixed in J2000, and then deconvolves the two jointly using clean. Finally the images are restored.

---

imager.predictcomp.html

imager.predictcomp - Function

2.3.1 Make a component list for a known object

Description

Make a component list for an object recognized by standard, one of setjy's flux density standards.

Arguments

Inputs		
objname	Name of the object	
	allowed:	string
	Default:	
standard	Name of the flux standard	
	allowed:	string
	Default:	
epoch	Time to use, as an epoch measure, e.g. me.epoch('UTC', '55555d'), for Solar System objects	
	allowed:	any
	Default:	variant 55555.0d
		epoch measure
freqs	The frequencies to use, in Hz	
	allowed:	doubleArray
	Default:	1.0e11
pfx	Prefix for the name of the component list	
	allowed:	string
	Default:	predictcomp

Returns

string

Example

```
clname = im.predictcomp('Ceres', 'Butler-JPL-Horizons 2010',
                        '2012-02-14/13:33:00', [3.45e11, 6.90e11], 'vd_')
```

This writes a component "list" named vd_spw0_Ceres_345GHz55971.6d.cl to disk containing a uniform disk component for Ceres as it is expected to appear at 345 and 690 GHz at 2012-02-14/13:33:00 UTC, and returns the name of the component list. Returns '' on error.

---

imager.makeimage.html

imager.makeimage - Function

2.3.1 Calculate images by gridding, etc.

Description

This tool function actually does gridding (and Fourier inversion if needed) of visibility data to make an image. It allows calculation of various types of image:

observed Make the dirty image from the DATA column (*default*)

model Make the dirty image from the MODEL_DATA column

corrected Make the dirty image from the CORRECTED_DATA column

residual Make the dirty image from the difference of the CORRECTED_DATA and MODEL_DATA columns

psf Make the point spread function

singledish Make a single dish image

coverage Make a single dish coverage image

holography Make a complex holography image

pb Make the primary beam as defined by setvp

Note the full `imager` equation is not used and so, for example, the primary beam correction is not performed. Use restore to get a residual image using the full `imager` equation where primary beam correction is performed.
A position shift can be applied when specifying the image parameters with defineimage. If a shift is specified then the uvw coordinates are reprojected prior to gridding, and a phase rotation is applied. If the image is a PSF then no phase shift is applied but the uvw are recomputed. To see the effects of the uvw reprojected, you can use the plotuv function.
If desired, the full complex image (before conversion to stokes I,Q,U,V) may be retained. Note that the image tool cannot load a complex image directly. Instead, use the imagecalc constructor to take *e.g.* the real and imaginary parts of the image.
For making single dish and holography images, the data are convolved onto the grid using a one of a number of options:

gridfunction='SF' Circularly symmetric prolate spheroidal wavefunction. This is always the same function in pixels. To get this to match to the antenna primary beam, the optimum cellsize to use in constructing the image is the antenna primary beam half-width-half-maximum times 1.20192.

gridfunction='BOX' Nearest neighbor gridding.

gridfunction='PB' The telescope primary beam is used as the convolution function. This function is the same in arcseconds, independent of the cellsize. This choice is optimum in the least squares sense. To override the default choice of telescope primary beam for a given telescope, use the function setvp. Usually the default will be acceptable.

To make a reasonable approximation to the sky, one should divide the type='singledish' image by the type='coverage' image, thresholding at some level. For example:

```
ia.open('scanweight');
 ia.statistics(s);
threshold = s.max / 10.0;
#
ia.imagecalc('sdimage',
      pixels=spaste('scanimage[scanweight>', threshold,
    ']/scanweight[scanweight>', threshold, ']'))
###ia.view(raster=True, axislabels=True);
```

Arguments

Inputs		
type	Type of output image	
	allowed:	string
	Default:	observed
image	Name of output image	
	allowed:	string
	Default:	
complexiimage	Name of output complex image	
	allowed:	string
	Default:	
verbose	Report things like the center frequency to the logger	
	allowed:	bool
	Default:	true
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
im.ft(model='3C273XC1.model', complist='3C273XC1.complist');
im.makeimage(type='residual', image='3C273XC1.residual')
im.makeimage(type='psf', image='3C273XC1.psf')
```

Fill in the MODEL_DATA column from Fourier transforming the model and the componentlist. Make the residual image and write it to 3C273XC1.residual.

Example

```
im.setvp(dovp=T, usedefaultvp=T, telescope='GBT');
im.makeimage(type='pb', image='gbt.pb')
```

In the above we may want to see what the primary beam we are using look like. May also be useful to deconvolve single dish images in the deconvolver tool.

---

imager.makemodelfromsd.html

imager.makemodelfromsd - Function

2.3.1 Make an initial model image from a Single Dish image

Description

This functions use an image from a single dish and make a model (clean component) image out of it. This allows one to use this as the starting model in a deconvoltion function e.g clean or mem This provides an alternative to feather. The difference between the two is that in `feather` the interferometer image is deconvolved first and the single dish image is put in at the end. Whereas if one starts with a model from the single dish image it will give a different starting point for the deconvolving algorithm to interpolate the missing short baseline.
The function setsdoptions may be used to set a factor by which to scale the SD image, if necessary.
The `sdpsf` parameter (optional) should be used if an external PSF image of the single dish is needed to calculate the beam parameters of the primary beam of the dish. This is usually needed if the dish image is from a non standard telescope or the beam is not in the `CASA` system.
The `mask` is a mask image that may be needed to be used for clean. This is usually the case when the dish image does not fully cover the field defined by defineimage.

Arguments

Inputs		
sdimage	Single Dish image	
	allowed:	string
	Default:	
modelimage	Name of output image to be used as model	
	allowed:	string
	Default:	
sdpsf	PSF of Single Dish if needed	
	allowed:	string
	Default:	
maskimage	mask image	
	allowed:	string
	Default:	

Returns
bool

Example

```
im.open('orion\_only.ms')
im.selectvis(field=range(10), spw=range(2))
im.defineimage(nx=1000, cellx='1arcsec', , phasecenter=4, spwid=[0,1])
im.setvp(dovp=T)
im.setoptions(ftmachine='mosaic')
im.setscales(nscales=3)
im.setsdoptions(scale=0.9);
im.makemodelfromsd(sdimage='orion\_gbt.im', modelimage='orion\_model', maskimage='orion.mask
im.clean(algorithm='mfmultiscale', model='orion\_model',
residual='orion.residual', image='orion.restored', gain=0.2, niter=500, mask='orion.mask')
```

```
In the above example we are making a mosaic with the fields 0 to 9. A
single dish image {\tt orion\_gbt.im} is used scaled down by a factor 0.9 to
make the initial model that is passed to multi-scale clean.
```

---

imager.mask.html

imager.mask - Function

2.3.1 Construct a mask image by thresholding an image

Description

A mask image is an image with the same shape as the other images but with values between 0.0 and 1.0 as a pixel value. Mask images are used in `imager` to control the region selected in a deconvolution. One makes a mask image by clipping the I part of the restored image (this function) or via the boxmask, regionmask, and exprmask functions. In this function, all points greater than the threshold are set to unity. The mask is the same in I,Q,U, and V. Note that exprmask is the most powerful method for making mask images.
In the Clark CLEAN, the mask image can usefully have any value between 0.0 and 1.0. Intermediate value discourage but do not rule out selection of clean components in that region. This is accomplished by multiplying the residual image by the mask prior to entering the minor cycle.
Note that if you do use a mask for the Clark or Hogbom Clean, it must cover only a quarter of the image. It is particularly important to check this when creating an image using a threshold. If it extends further, the easiest fix is to use getchunk and getchunk to set parts of it to zero.

Arguments

Inputs		
image	name of template image	
	allowed:	string
	Default:	
mask	name of mask image	
	allowed:	string
	Default:	
threshold	threshold for mask	
	allowed:	any
	Default:	variant 0.0Jy
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
im.mask( image='bigimage', mask='bigmask',threshold='0.07Jy')
im.clean(mask='bigmask', model='3C273XC1.clean.masked', niter=1000)


Makes the image bigmask, and then sets it to unity
for all points where the Stokes I in bigimage is
greater than 0.07. Then clean using it as the mask.
```

---

imager.mem.html

imager.mem - Function

2.3.1 Calculate a deconvolved image with selected mem (maximum entropy) algorithm

Description

Makes a mem image using either the Cornwell-Evans maximum entropy or maximum emptiness algorithms, using the single field or multi-field contexts. The maximum entropy algorithm is the default. The mem is performed on the residual image calculated from the visibility data currently selected. Hence the first step performed in mem is to transform the current model or models (optionally including a componentlist) to fill in the MODEL_DATA column, and then inverse transform the residual visibilities to get a residual image. This residual image is then deconvolved using the corresponding point spread function. This means that the initial model is used as the starting point for the deconvolution. Thus if you want to restart a mem, simply set the model to the model that was previously produced by clean.
Mask images are used to constrain the region that is to be deconvolved. To make mask images, use either boxmask (to define a mask via the corner locations blc and trc) or mask (to define a mask via thresholding an existing image). The default mask is the inner quarter of the image.
The MEM deconvolution only operates on one Stokes parameter at a time. Joint MEM deconvolution for multiple Stokes parameters will be implemented in the future.
Some reference regarding MEM : Cornwell and Evans, Astronomy and Astrophysics (ISSN 0004-6361), vol. 143, no. 1, Feb. 1985, p. 77-83.
Narayan and Nityananda, Annual review of astronomy and astrophysics. Volume 24 (A87-26730 10-90). Palo Alto, CA, Annual Reviews, Inc., 1986, p. 127-170.
The mem algorithms possible are:

Cornwell-Evans Maximum Entropy (entropy) The classic "vm" or "vtess" deconvolution algorithm.

Cornwell-Evans Maximum Emptiness (emptiness) The historic, but largely undocumented, modification to the Cornwell-Evans algorithm which seeks a model image which is consistent with the data and simultaneously minimizes the number of pixels with no emission (meaning "with pixel values below the noise level").

Multi-field Maximum Entropy (mfentropy) Deconvolution is split into minor and major cycles. For each field, the MEM analog of a Clark Clean

minor cycle is performed. In the major cycle, the emission thus modelled is subtracted either from the original visibilities (for multiple fields) or using a convolution (for only one field). The latter is much faster.

Multi-field Maximum Emptiness (mfemptiness) Just like `mfentropy`, but with emptiness.

The multi-field mem (`mfentropy` or `mfemptiness`) should be used if either of two conditions hold:

1. Multiple fields are to be deconvolved simultaneously **OR**
2. Primary beam correction is enabled. In this case, a mosaiced mem is performed.

Note that for the single pointing algorithms, only a quarter of the image may be deconvolved. If no mask is set, then the deconvolved region defaults to the inner quarter. If a mask larger than a quarter of the image is set, then only the quarter starting at the bottom left corner is used. However, for the multi-field imaging, the entire field may be imaged because the major cycles either do an exact subtraction from the visibilities or because PSF extent is more than twice the extent of the primary beam support.
Before `mem` can be run, you must run `selectvis` and `defineimage`. Before `mem` can be run with a multi-field algorithm, you should run `setvp`. You may want to run `setmfcontrol` before running `mem` with a multi-field algorithm, though the default control values may be acceptable.

Arguments

Inputs		
algorithm	Algorithm to use	
	allowed:	string
	Default:	entropy
		emptiness
		mfentropy
		mfemptiness
		entropy
		entropy
niter	Number of Iterations	
	allowed:	int
	Default:	20
sigma	Image sigma to try to achieve	
	allowed:	any
	Default:	variant 0.001Jy
targetflux	Target flux for final image	
	allowed:	any
	Default:	variant 1.0Jy
constrainflux	Constrain image to match target flux? else targetflux used only to initialize model	
	allowed:	bool
	Default:	false
displayprogress	Display the progress of the cleaning?	
	allowed:	bool
	Default:	false
model	Names of model images	
	allowed:	stringArray
	Default:	
keepfixed	Keep model fixed	
	allowed:	boolArray
	Default:	false
complist	Name of component list	
	allowed:	string
	Default:	
prior	Names of mem prior images	
	allowed:	stringArray
	Default:	
mask	Names of mask images (0=>no emission, 1=>emission permitted	
	allowed:	stringArray
	Default:	
image	Names of restored images	
	allowed:	stringArray
	Default:	
residual	Names of residual images	
	allowed:	stringArray
	Default:	
async	Run asynchronously in the background?	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
im.mem(model='3C273XC1.mem.model',
mask='3C283XC1.mask', niter=40, sigma='0.001Jy')
```

---

imager.nnls.html

imager.nnls - Function

2.3.1 Calculate a deconvolved image using the NNLS algorithm

Description

Solve for the model brightness using the Briggs' Non-Negative Least Squares algorithm. Since NNLS works only on the I image, the I pixels in the current image is set to zero where the fluxmask is > 0.0, then NNLS is used to estimate the I-pixels for that region. The deconvolution is performed on the residual image calculated from the visibility data currently selected. Hence the first step performed in clean is to transform the current model to fill in the MODEL_DATA column, and then inverse transform the residual visibilities to get a residual image. This residual image is then deconvolved using the corresponding point spread function.
Some other points to remember are that rather than explicit boxes, mask images are used to constrain the region that is to be deconvolved. For NNLS, there are two masks, the fluxmask specifying the region within which flux is allowed, and the datamask specifying the region of the dirty image to be used as constraints. Typically the datamask will be somewhat larger than the fluxmask. On a large machine, a practical limit to both will be about 5000-6000 pixels. Hence NNLS is only useful for compact tools. (For more details, see the Briggs thesis). To make mask images, use either boxmask (to define a mask via the corner locations blc and trc) or mask (to define a mask via thresholding an existing image).
On the canonical CASA machine with 64MBytes of physical memory, you should try to keep the product of the pixels in the fluxmask and the datamask below about 5-10 million. Otherwise the solution phase will swap badly.

Arguments

Inputs		
model	Name of image	
	allowed:	stringArray
	Default:	
keepfixed	Keep model fixed	
	allowed:	boolArray
	Default:	false
complist	Name of component list	
	allowed:	string
	Default:	
niter	Number of Iterations, set to zero for no NNLS	
	allowed:	int
	Default:	0
tolerance	Tolerance for solution	
	allowed:	double
	Default:	1e-06
fluxmask	Name of mask for allowed flux	
	allowed:	stringArray
	Default:	
datamask	Name of mask for constraint pixels in dirty image	
	allowed:	stringArray
	Default:	
image	Names of restored images	
	allowed:	stringArray
	Default:	
residual	Names of restored images	
	allowed:	stringArray
	Default:	
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
im.nnls(image='3C273XC1.nnls.image', model='3C273XC1.nnls.model',
fluxmask='3C283XC1.fluxmask', datamask='3C273XC1.datamask', niter=1000,
tolerance=0.00001)
```

imager.open.html

imager.open - Function

2.3.1 Open a new MeasurementSet, for processing, closing current MeasurementSet

Description

Close the current MeasurementSet and open a new MeasurementSet instead. The current state of `imager` is retained, except for the data selection.

Arguments

Inputs		
thems	New MeasurementSet to be processed	
	allowed:	string
	Default:	
compress	Compress calibration columns?	
	allowed:	bool
	Default:	false
usescratch	If true: Imager will use corrected data column and make scratch columns of they donot exist	
	allowed:	bool
	Default:	false

Returns

bool

---

imager.pb.html

imager.pb - Function

2.3.1 Applies or corrects for a primary beam

Description

Multiply (`operation='apply'`) or divide (`operation='correct'`) by the primary beam function. The primary beam can be applied to images and/or Componentlists.
If `pointingcenter==false` then you must specify `inimage` and the pointing center is taken from its reference direction. Otherwise, `pointingcenter` must be a Direction measure. It cannot take on the value `True`.
The applied primary beam function is deterimed as follows. If you used function Imager.setvp to set an external voltage pattern table, then this is where the applied primary beam will come from (regardless of whether you set `inimage` or not). If you did not run this function, then you must supply argument `inimage`. The telescope name embedded in its Coordinate System will be used to determine the primary beam function.

Arguments

Inputs		
inimage	Input image to apply beam to	
	allowed:	string
	Default:	
outimage	Output image after beam is applied	
	allowed:	string
	Default:	
incomps	Input Componentlist table name	
	allowed:	string
	Default:	
outcomps	Output Componentlist table name	
	allowed:	string
	Default:	
operation	Operation	
	allowed:	string
	Default:	correct
		apply
pointingcenter	Pointing center for primary beam application: default N.Pole	
	allowed:	any
	Default:	variant
parangle	Parallactic angle for calculation	
	allowed:	any
	Default:	variant 0.0deg
pborvp	Primary Beam or Voltage Pattern	
	allowed:	string
	Default:	vb
		pb
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
# make a flat image
im.make('flat.image');
```

```
ia.open('flat.image');
arr=ia.getchunk();
arr[0:len(arr), 0:len(arr[0])] = 1.0;
ia.putchunk(arr);
ia.done()
arr = false;
#
# as we are using "pointingcenter=F", it defaults to the image center
im.pb(inimage='flat.image', outimage='pb.image', pointingcenter=F)
```

---

imager.plotsummary.html

imager.plotsummary - Function

2.3.1 Plot a summary of field and spectral window ids

Description

Performs a simple plot of the field and spectral window IDs versus time (after sorting).

Arguments

Returns
bool

Example

```
m = fitstoms('3C273XC1.ms', '3C273XC1.fits'); m.close()
im.open('3C273XC1.ms')
im.plotsummary()
```

---

imager.plotuv.html

imager.plotuv - Function

2.3.1 Plot the uv coverage

Description

Performs a simple plot of the uv coverage of all selected data.
Optionally, plotuv will rotate the uvw coordinates to the specified phase center (set via defineimage).

Arguments

Inputs		
rotate	Rotate uvw coordinates to specified phase center?	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
im.open('3C273XC1.ms')
im.plotuv(false)
```

---

imager.plotvis.html

imager.plotvis - Function

2.3.1 Plot the visibility amplitudes as a function of u-v radius (also, see visplot tool

Description

Performs a simple plot of the visibility amplitudes of all selected data.

Arguments

Inputs		
type	Type of plot: can contain all, observed, corrected, model, residual	
	allowed:	string
	Default:	all
		observed
		corrected
		model
		residual
		all
increment	Increment in points to plot	
	allowed:	int
	Default:	1

Returns

bool

Example

```
im.open('3C273XC1.ms')
im.plotvis(increment=10)
```

imager.plotweights.html

imager.plotweights - Function

2.3.1 Plot the visibility weights as a function of u-v radius

Description

Performs a plot of the visibility weights of all selected data (stored in the IMAGING_WEIGHT column of the MeasurementSet). The plot can be of the gridded weights (type='gridded') or ungridded.

Arguments

Inputs		
gridded	Do gridded plot?	
	allowed:	bool
	Default:	false
increment	Increment in points to plot	
	allowed:	int
	Default:	1

Returns

bool

Example

```
im.open('3C273XC1.ms')
im.defineimage(cellx='0.7arcsec', celly='0.7arcsec')
im.weight('briggs')
im.plotweights(gridded=True,increment=10)
```

imager.regionmask.html

imager.regionmask - Function

2.3.1 Construct a mask image from a region

Description

A mask image is an image with the same shape as the other images but with values between 0.0 and 1.0 as a pixel value. Mask images are used in imager to control the region selected in a deconvolution.
In the Clark CLEAN, the mask image can usefully have any value between 0.0 and 1.0. Intermediate value is discouraged but do not rule out selection of clean components in that region. This is accomplished by multiplying the residual image by the mask prior to entering the minor cycle. Note that if you do use a mask for the Clark or Hogbom Clean, it must cover only a quarter of the image. `regionmask` does not enforce this requirement.
The function regionmask also allows multiple regions to be used. A record of the regions can be made as in the example below.
Regions can be made in many different ways using the regionmanager functions. An example using wbox function is given below. The default regionmanager tool 'rg' can be used for cases the user want to have flexibility in manipulating regions. The `region` parameter takes a record that comes from the regionmanager output. The parameter boxes allow the user to sent in a list of 4 elements numbers representing blc's and trc's
If both the parameters, `regions` and `boxes` are used the a union is done with the two sets of region thus defined.

Arguments

Inputs		
mask	name of mask image	
	allowed:	string
	Default:	
region	Region record usually from regionmanager	
	allowed:	record
	Default:	unset
boxes	list of 4 elements lists e.g [[xblc1, yblc1, xtrc1, ytrc1], [[xblc2, yblc2, xtrc2, ytrc2]]	
	allowed:	any
	Default:	variant
circles	list of 3 elements lists e.g [[rad0, xcen0, ycen0], [rad1,xcen1, ycen1], ...]	
	allowed:	any
	Default:	variant
value	Value to set the mask to	
	allowed:	double
	Default:	1.0

Returns
bool

Example

```
Makes a mask  then cleans using it.

im.open('test.ms')
im.selectvis(field=0, spw=0)
im.defineimage(nx=400, cellx='0.001arcsec', phasecenter=0)
a=[100.0, 100.0, 200, 200.0]
b=[50, 50, 80, 80]
im.regionmask(mask='bigmask', boxes=[a,b])
im.clean(mask='bigmask', model='3C273XC1.clean.masked', niter=1000)

Another example using rg.wbox function:
ia.open('dirty')
cs = ia.coordsys()
```

```
rg.setcoordinates(cs.record())
r1 = dg.wbox(blc=['173pix', '347pix'], trc=['183pix', '370pix'])
im.regionmask(mask='bigmask',region=r1)
```

Or using a dict of regions:

```
r2=rg.wbox(blc=['180pix', '344pix'], trc=['191pix', '369pix'])
r3=rg.wbox(blc=['189pix', '341pix'], trc=['204pix', '364pix'])
regs={"reg1"':r1, "reg2":r2, "reg3":r3}
rec=rg.makeunion(regs)
im.regionmask(mask='bigmask',region=rec)
```

If quantities are to be used to define regions the following is a an example

```
im.regionmask(mask='joetest',boxes=['15:23:32.902','+05.19.32.089','15:22:28.631','+05.28.52
```

---

imager.regiontoimagemask.html

imager.regiontoimagemask - Function

2.3.1 union a mask image with various regions

Description

This function is very similar to regionmask function except that the mask image has to be existant already and this is an independent helper function (i.e does not care about the state of the imager tool... e.g does not need imager to have an attached ms).

Arguments

Inputs		
mask	name of mask image	
	allowed:	string
	Default:	
region	Region record usually from regionmanager	
	allowed:	record
	Default:	unset
boxes	list of 4 elements lists e.g [[xblc1, yblc1, xtrc1, ytrc1], [[xblc2, yblc2, xtrc2, ytrc2]]	
	allowed:	any
	Default:	variant
circles	list of 3 elements lists e.g [[rad0, xcen0, ycen0], [rad1,xcen1, ycen1], ...]	
	allowed:	any
	Default:	variant
value	Value to set the mask to	
	allowed:	double
	Default:	1.0

Returns

bool

Example

```
Makes a mask  then cleans using it.

a=[100.0, 100.0, 200, 200.0]
b=[50, 50, 80, 80]
im.regiontoimagemask(mask='bigmask', boxes=[a,b])
im.clean(mask='bigmask', model='3C273XC1.clean.masked', niter=1000)

Another example using rg.wbox function:
ia.open('dirty')
cs = ia.coordsys()
rg.setcoordinates(cs.record())
r1 = dg.wbox(blc=['173pix', '347pix'], trc=['183pix', '370pix'])
im.regionmask(mask='bigmask',region=r1)

Or using a dict of regions:

r2=rg.wbox(blc=['180pix', '344pix'], trc=['191pix', '369pix'])
r3=rg.wbox(blc=['189pix', '341pix'], trc=['204pix', '364pix'])
regs={"reg1"':r1, "reg2":r2, "reg3":r3}
rec=rg.makeunion(regs)
im.regionmask(mask='bigmask',region=rec)

If quantities are to be used to define regions the following is a an example

im.regionmask(mask='joetest',boxes=['15:23:32.902','+05.19.32.089','15:22:28.631','+05.28.52
```

imager.residual.html

imager.residual - Function

2.3.1 Calculate the residual image with respect to current model and component list

Description

Calculate the residuals corresponding to the model and componentlist. *Note that the model visibilities are updated.*

Arguments

Inputs		
model	Names of input models	
	allowed:	stringArray
	Default:	
complist	Name of component list	
	allowed:	string
	Default:	
image	Names of output residual images	
	allowed:	stringArray
	Default:	
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
- im.residual(model='3C273XC1.clean', complist='3C273XC1.cl',
image='3C273XC1.clean.residual')
```

imager.restore.html

imager.restore - Function

2.3.1 Calculate the restored image with restored model, component list, and residuals

Description

Restore the residuals to a smoothed version of the model. The model images are convolved with the specified Gaussian beam and then the residual images are added. *Note that the model visibilities are updated and thus reflect the model and componentlist that was used.*. Use setbeam to set the beam parameters.

Arguments

Inputs		
model	Names of input model	
	allowed:	stringArray
	Default:	
complist	Name of component list	
	allowed:	string
	Default:	
image	Names of output restored images	
	allowed:	stringArray
	Default:	
residual	Names of residual images	
	allowed:	stringArray
	Default:	
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
- im.setbeam(bmaj='2.0arcsec', bmin='2.0arcsec')
- im.restore(model='3C273XC1.clean', image='3C273XC1.clean.restored',
```

---

imager.updateresidual.html

imager.updateresidual - Function

2.3.1 Calculate the residual and restored images with new modified model, component list,

Description

This function is for efficiency and speed purpose only. Same as restore It is to be used after you have used clean or mem ...but you wish to tweak the model image, say by clipping unwanted components and it will avoid unnecessary recalculating of psf but will do a proper prediction of the new model visibilities and recalculate residual and restored images.

Arguments

Inputs		
model	Names of input model	
	allowed:	stringArray
	Default:	
complist	Name of component list	
	allowed:	string
	Default:	
image	Names of output restored images	
	allowed:	stringArray
	Default:	
residual	Names of residual images	
	allowed:	stringArray
	Default:	

Returns
bool

Example

```
- im.setbeam(bmaj='2.0arcsec', bmin='2.0arcsec')
```

```
- im.restore(model='3C273XC1.clean', image='3C273XC1.clean.restored',
```

---

imager.sensitivity.html

imager.sensitivity - Function

2.3.1 Calculate rms sensitivity

Description

NB: The implementation in this function will be removed for CASA v4.5. We now recommend that the im.apparentsens() function be used instead of this one, especially if their weights are initialized and calibrated.
Calculate the point source sensitivity for the selected data, both absolutely and relatively (to that for natural weighting).
To do the calculation, we use the imaging weights (in the column called IMAGING_WEIGHT) as calculated from the WEIGHT column, and an estimate of the effective net bandwidth and integration time. The calculation therefore includes all the effects of weight and filter.
The output is an array with mixed elements. Counting from zero, the second element (out[1]) is the net sensitivity, third element is the ratio of the reduction in sensitivity due to the chosen weighting scheme. This ratio is 1.0 for Natural weight and greater than one for all other weighting schemes. (NOTE: Further testing is required of this value and hence this is kept separate for now).
The sensitivity calculations require Tsys and collecting area of the antenna. These quantities are not known from the MS. The sensitivity is therefore returned in units of Jy m$\hat{2}$/K. Multiplying the second elements with the ration of the Tsys and effective antenna collecting area will give the sensitivity in Jy/beam units.
The fourth elements of the return value is a record with the following keys: 'nbaselines', 'effectiveintegration', 'effectivebandwidth', 'sumwt' and 'spwid'. These can be used to get the number of baselines used, effective integration time (in sec), the effective bandwidth (in Hz), the sum of weights and the absolute spectral window IDs used.

Arguments

Outputs		
pointsource	Calculated point source sensitivity (Jy m$\hat{2}$ / (K beam))	
	allowed:	record
	Default:	
relative	Calculated relative sensitivity	
	allowed:	double
	Default:	
sumweights	Calculated sum of weights	
	allowed:	double
	Default:	
senrec	Record per SPW per chan sensitivity calculations	
	allowed:	record
	Default:	
Inputs		
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
a=im.sensitivity(false);
print 'Sensitivity =', a[1];
print 'Relative to Natural Weighting = ', a[2];
```

imager.apparentsens.html

imager.apparentsens - Function

2.3.1 Calculate rms sensitivity directly from weights

Description

This function calculates the point source sensitivity for the data selected by im.selectvis(...), and according to the imaging weighting parameters specified in im.weight(...) and im.defineimage(...). The calculation is performed solely using the weight information stored in the MS WEIGHT column (WEIGHT_SPECTRUM tbd), and as adjusted by the net imaging weighting function (natural, uniform, robust, taper, etc.). Therefore, it is assumed that the MS WEIGHTs have been properly initialized and calibrated along with the visibility data. As long as the WEIGHTs are in the inverse square units of the visibilities (i.e., inverse variance weights), the calculation should yield the real theoretical imaging sensitivity for data at any stage of the calibration (though data at early and intermediate stages of calibration may not be sufficiently coherent for imaging at high–or even modest–fidelity).
Two values are reported in the logger and returned (see example below). First, the apparent sensitivity (in the units implied by the WEIGHTs' units), for the specified imaging weighting scheme. Second, a unitless factor describing the ratio of the apparent sensitivity to that obtained with pure 'natural' weighting (the nominal peak sensitivity). When 'natural' weighting is selected, this ratio factor will be 1.0; all other weighting choices will yield an apparent sensitivity ratio greater than 1.0.
Currently, this function reports only the continuum sensitivity for the selected data, and in particular, for the aggregate bandwidth indicated by the spectral window selection. The calculation further assumes that the visibility samples are each entirely independent (i.e., no redundant samples such as would occur for overlapping spectral windows).
A future version of this function will support reporting a sensitivity spectrum for the spectral line case (including support for WEIGHT_SPECTRUM). For now, spectral line sensitivity may be reasonably estimated by dividing the reported sensitivity by the square root of the fractional bandwidth of a single image channel, or by selecting a bandwidth matching the width of a single image channel.

Arguments

Outputs	
pointsource	Calculated apparent point source sensitivity (in units implied by the MS weights) allowed: double Default:
relative	Ratio of apparent sensitivity relative to natural weighting allowed: double Default:
Inputs	
async	Run asynchronously in the background allowed: bool Default: false

Returns
bool

Example

```
# open and set up selection and image plane parameters
im.open('mydata.ms')
im.selectvis(field='2',spw='0')
im.defineimage(mode='mfs',spw=0,stokes='I',cellx='15arcsec',celly='15arcsec',nx=256,ny=256)

# report natural weighting sensitivity
im.weight(type='natural')
nat=im.apparentsens();
print 'Natural Sensitivity =', nat[1];
print 'Relative to Natural Weighting = ', nat[2];

# switch to uniform weighting
im.weight(type='uniform')
uni=im.apparentsens();
print 'Uniform Sensitivity =', uni[1];
print 'Relative to Natural Weighting = ', uni[2];

# switch to briggs weighting
im.weight(type='briggs',robust=0.0)
rob=im.apparentsens();
print 'Briggs Sensitivity =', rob[1];
```

```
print 'Relative to Natural Weighting = ', rob[2];

im.close()
```

---

imager.setbeam.html

imager.setbeam - Function

2.3.1 Set the beam parameters for clean restoration

Description

This sets the clean beam that will be used in all restoration operations.

Arguments

Inputs		
bmaj	Major axis of beam	
	allowed:	any
	Default:	variant 1.0arcsec
bmin	Minor axis of beam	
	allowed:	any
	Default:	variant 1.0arcsec
bpa	Position angle of beam	
	allowed:	any
	Default:	variant 0deg
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns

bool

---

imager.selectvis.html

imager.selectvis - Function

2.3.1 Select visibilities for subsequent processing

Description

This setup tool function selects which data are to be used subsequently. After invocation of selectvis, only the selected data are operated on. Thus, for example, in imaging, only the selected data are gridded into an image, and in plotting, only the selected data are plotted.
Data can be selected by field and spectral window ids. Note that all data thus selected are passed to imaging, and may or may not be imaged, depending on how the image was constructed using defineimage. For example, in mosaicing, use fieldid in defineimage to control what pointing is used to define the field center, and use fieldid in selectvis to control what pointings are used in the imaging.
For spectral processing, it is possible to make cubes out multi-spectral window selections but the selection and combination can be a bit confusing (any hint at how to make it clearer is welcome).
If the default values are not used, then data to be used can be selected channel wise. The

nchan is the number of data channels selected. It defaults to -1 (interpreted as all channels).

start is the first channel from input dataset that is to be used. It defaults to 0 (i.e. first channel).

step gives the increment between selected input channels. It defaults to 1 channel. A value of `n` means that `n-1` data channels will not not be used.

By choosing the parameters for selectvis and defineimage correctly, one may obtain various mappings of visibility channels to image channels. For example, to average 512 visibility channels into 64 image channels (producing image channels consisting of 8 visibility channels):
im.defineimage(mode='channel', spw=0, nchan=64, start=1, step=8);
im.selectvis(spw=0, nchan=512, start=1, step=1) im.clean(.....);
This averages the spectral channels during the gridding process. If one wanted to only include every 8th channel in the deconvolution, one would do:
im.selectvis(nchan=64, start=1, step=8) im.defineimage(mode='channel', nchan=64, start=1, step=8); im.clean(....);
For velocity and opticalvelocity modes, the mstart and mstep are the start and step velocities as strings.

im.defineimage(mode='velocity', nchan=64, start='20 km/s', step='-100m/s');
im.selectvis(spwid=[-1]); ###selecting all data spectral windows im.clean(...);
If the image and data selections differ, then averaging is done during the gridding and degridding process in the image deconvolution.
im.defineimage(mode='channel', nchan=64, start=1, step=8);
im.selectvis(nchan=512, start=1, step=1) im.clean()
Note: The channels numbers used in `defineimage` and `selectvis` refers to the same channel. So if a channel is not selected in `selectvis` but is selected in `defineimage`, then blank channels image are made. The example below will result in the having the first 6 (0-5) channels in the image to be blank.
im.selectvis(nchan=50, start=6, step=1) #selected chan 6-55
im.defineimage(mode='channel', nchan=50, start=0, step=1);
# will try to image channel 1-50. But as previously only channel 6-55 # was selected only channel 6-50 will have data; images of channels # 1-5 are blank
im.clean(....)
For multi-spectral window cube imaging the selection of the data can be done as follows
im.selectvis(nchan=[50,60], start=[0,0], step=[1,1], spw=[0,1])
im.defineimage(mode='channel', nchan=110, start=0, step=1, spw=[0,1]);
The above means that you would make a data selection of 50 channels (starting from 0 steping 1) from the first spectral window and 60 channels (starting from 1 steping 1). The defineimage defines the image to be a cube of 110 channels. The caveat is the step size in the frequency direction is the step size of the first spectral window. If the step size of channels of the two spectral windows are different then one is better off defining the image cube in velocities (e.g. as below).
im.selectvis(nchan=[50,60], start=[0,0], step=[1,1], spw=[1,2])
im.defineimage(mode='velocity', nchan=200, mstart='20km/s',
mstep='-100m/s');

Arguments

Inputs	
vis	Measurementset for which this selection applies; an empty string "" implies that it is to be applied in ms used in open allowed: string Default:
nchan	Number of channels to select allowed: intArray Default: -1
start	Start channels (0-relative) allowed: intArray Default: 0
step	Step in channel number allowed: intArray Default: 1
spw	Spectral Window Ids (0 relative) to select; -1 interpreted as all allowed: any Default: variant -1
field	Field Ids (0 relative) or Field names (msselection syntax and wilcards are used) to select allowed: any Default: variant -1
baseline	Antenna Ids (0 relative) or Antenna names (msselection syntax and wilcards are used) to select allowed: any Default: variant -1
time	Limit data selected to be within a given time range. The syntax is defined in the msselection link allowed: any Default: variant
scan	Limit data selected on scan numbers. The syntax is defined in the msselection link allowed: any Default: variant
intent	Limit data selected on observation intent. The syntax is defined in the msselection link allowed: string Default:
observation	Limit data using observation IDs. The syntax is defined in the msselection link allowed: any Default: variant
uvrange	Limit data selected on uv distance. The syntax is defined in the msselection link allowed: any Default: variant
taql	For the TAQL experts, flexible data selection using the TAQL syntax allowed: string Default:
usescratch	If True: imager will use CORRECTED_DATA column

Returns
bool

Example

```
im.open('3C273XC1.MS');
im.selectvis(nchan=512,start=1,step=1, taql='SCAN_NUMBER > 10 && FIELD_ID==2')


Time range selection

im.selectvis(field=range(0,10), time='> 2000/09/21/12:00:00')


select some antennas, for all fields that begins with {\tt 'ngc'}

im.selectvis(field='ngc*', baseline=[0, 10, 20])


And for those that the standard parameters are not flexible enough
there is the taql parameter. This for people who knows the different
columns of the MeasurementSet

im.selectvis(taql="ANTENNA1==0 && ANTENNA2==3")


Imager allows the user to make an image from multiple ms, without the
need to concatenate them. To do this then the im.open method should
{\bf not} be used at all but multiple calls of selectvis with the
parameter vis pointing to each ms should be used. The other
parameters can be used to make selection on each ms


im.selectvis(vis='myms1.ms', field=0, spw=[0,1], nchan=[40, 50], start=[5,10])
im.selectvis(vis='myms2.ms', field=10, spw=[2], nchan=[40], start=[5])
im.selectvis(vis='myms3.ms', field=range(0,10), time='> 2002/10/15/20:30:45')
```

imager.setjy.html

imager.setjy - Function

2.3.1 Compute the model visibility for a specified source flux density

Description

Compute the model visibility for a specified source flux density, and insert into the MODEL_DATA column. The source flux density for a set of standard flux density reference sources may optionally be pre-computed, by setting the input flux density to -1 (the default). At present, these include 3C286, 3C48, 3C147, 3C138, and 1934-638. In this case, if the source is not in this set, an unpolarized flux density of 1 Jy will be assumed. Users may also specify `standard='SOURCE'` to use the model(s) in the SOURCE_MODEL column of the SOURCE subtable.
Users may also specify a model image that will be scaled to the specified total flux density (or that computed for reference sources). When a model image is specified, setjy will only permit processing one field, and will currently only process Stokes I.

Arguments

Inputs		
field	Field Id (0-relative) or name	
	allowed:	any
	Default:	variant
spw	Spectral Window Id. (0-relative)	
	allowed:	any
	Default:	variant
modimage	A model image	
	allowed:	string
	Default:	
fluxdensity	Specified flux density (I,Q,U,V) in Jy (lookup the value; use 1.0 if not found)	
	allowed:	doubleArray
	Default:	0.0 0.0 0.0 0.0
standard	Flux density standard	
	allowed:	string
	Default:	Baars
		Perley 90
		Perley-Taylor 95
		Perley-Taylor 99
		Perley-Butler 2010
		SOURCE
scalebychan	Do the flux scaling on a per channel basis else on a spw basis. Effectively True if fluxdensity is specified.	
	allowed:	bool
	Default:	false
spix	Spectral index for fluxdensity. S = fluxdensity * (freq/reffreq)**spix	
	allowed:	doubleArray
	Default:	0.0
reffreq	Reference frequency for spix.	
	allowed:	any
	Default:	variant 1GHz
polindex	Coefficients for Taylor expansion of Polarization index	
	allowed:	doubleArray
	Default:	0.0
polangle	Coefficients for Taylor expansion of Polarization angle in radians	
	allowed:	doubleArray
	Default:	0.0
rotmeas	rotation measure (rad/lambda**2)	
	allowed:	double
	Default:	0.0
time	Time range to operate on	
	allowed:	string
	Default:	
scan	Scan(s) to operate on	
	allowed:	string
	Default:	
intent	Observation intent	
	allowed:	string
	Default:	
observation	Observation ID(s) to operate on	

Returns
record

Example

```
im.setjy(fieldid=2, spwid=-1, fluxdensity=[2.6,0.2,0.3,0.5],standard='Baars')


Compute the model visibility for field id. 2 to the specified
point-source (I,Q,U,V) for all spectral windows id.'s on the
Baars flux density scale.
```

---

imager.ssoflux.html

imager.ssoflux - Function

2.3.1 Use setjy instead.

Description

This was an experimental clone of setjy while flux calibration with Solar System objects was being tested. It has been merged back into setjy.*

Arguments

Inputs

Returns
bool

Example

---

imager.setmfcontrol.html

imager.setmfcontrol - Function

2.3.1 Set various cycle control parameters for multi-field and wide-field imageing.

Description

Control parameters for mosaicing or wide-field imaging which are not required in single field deconvolution are set here to streamline the user interface. As multifield and widefield imaging is accomplished by deconvolution in cycles, many of these parameters control how the deconvolution cycles are ended.

cyclefactor: this parameter helps in lowering or increasing the threshold at which the deconvolution cycle will stop and degrid and subtract from the visibilities. For very bad PSFs you may want to reconcile with the visibilties often, thus a larger number is required here...(4 to 5). For very well behaved data you may want to deconvolve deep before reconciling: a lower number is used (1.5 to 2.0).

cyclespeedup: this is used if the PSF is not well behaved and you want clean to raise by 2 the threshold if it has not reached the threshold in this number of iteration

cyclemaxpsffraction: similar to cyclefactor, but this is an explicit fraction of the PSF peak. The final threshold is computed using min(cyclemaxpsffraction, cyclefactor * maxPSFsidelobe). Valid values are between 0.0 and 1.0.

stoplargenegatives: This parameter is exclusively for when using multiscale clean. This is used to stop the component search when the largest scale has found this number of negative components. -1 here means that continue component search even if the largest component is negative.

stoppointmode: Again exclusively for when using multiscale clean. The clean will stop if the smallest scale receives this number of consecutive components.

minpb: This is to defined up to what level the voltage pattern is going to applied when using setvp. The default is 0.1 of the primary beam or the voltage pattern defined for the antenna.

scaletype: This parameter cab be NONE or SAULT. If NONE the image is not scaled, if SAULT is used the image is weighted so that the noise is

kept uniform across the image. The next two parameters defines how the SAULT weighting is limited. Obviously then the flux scale is not uniform across the image. To get the right flux multiply the image with the fluxscale image.

constpb: this parameter defines up to what amplitude of the Primary beam the noise floor is kept uniform, when using SAULT as scaletype.

fluxscale: use this to give a filename to store the factor image to apply to the image to get the fluxscale right.

flatnoise: (default true) Set to false if you want clean components for mosaic to be searched in the residual image that is effectively multiplied by the $beam^2$. This means when the noise is determined after the antenna, searching in the optimum domain of $signal/(sigma^2)$. For meter wavelengths where noise is determined by the sky, it is no longer optimal.

Arguments

Inputs	
cyclefactor	Cycle threshold = max_resid * min(cyclemaxpsffraction, this * max_sidelobe) allowed: double Default: 1.5
cyclespeedup	Cycle threshold doubles in this number of iterations allowed: double Default: -1
cyclemaxpsffraction	Cycle threshold = max_resid * min(this, cyclefactor * max_sidelobe) allowed: double Default: 0.8
stoplargenegatives	Stop the multiscale cycle for the first n cycles when a negative comp is found on the largest scale allowed: int Default: 2
stoppointmode	Stop multiscale altogether if the smallest scale recieves this many consecutive components allowed: int Default: -1
minpb	Minimum PB level to use allowed: double Default: 0.1
scaletype	Image plane flux scale type allowed: string Default: SAULT NONE NONE
constpb	In Sault weighting the flux scale is constant above this PB level allowed: double Default: 0.4
fluxscale	Names of flux scale images for mosaicing allowed: stringArray Default:
flatnoise	Set to false if clean component search is to be done in an optimal signal/noise residual image if true will clean in a constant noise image allowed: bool Default: true

Returns
bool

Example

```
im.setmfcontrol(cyclefactor=2.0, cyclespeedup=niter/10, cyclemaxpsffraction=0.8,
stoplargenegatives=T, stoppointmode=10, fluxscale='image.fluxscale');
```

---

imager.setoptions.html

imager.setoptions - Function

2.3.1 Set some general options for subsequent processing

Description

This function is for setting different gridding and memory options

ftmachine The options for ftmachine are:

- **ft** Standard interferometric gridding
- **sd** Standard single dish gridding
- **both** ft and sd as appropriate.
- **wproject** option for using the wproject algorithm for wide-field imaging; when this option is used the parameter `wprojplanes` define the number of convolution functions to be used
- **mosaic** option to use the gridder that uses the primary beam as the convolution function in gridding

cache The size of the cache used (in complex pixels) during the gridding process. The default is to use half the physical memory of the machine as specified by the aipsrc variable system.resources.memory.

tile The side of the tile (in complex pixels) during the gridding process.

gridfunction The gridding function used. Currently only Box-car ('BOX') and Prolate Spheriodal Wave Function ('SF') are supported. In the case of Single-Dish imaging the Primary Beam ('PB'), Gaussian ('GAUSS'), and Gaussian * Jinc ('GJINC') also can be used.

location For some unusual types of image, one needs to know the location to be used in calculating phase rotations. For example, one can specify images to be constructed in azel, in which case, an antenna position must be chosen. One can use functions of measures: either observatory to get the position of a named observatory (*e.g.* me.observatory('ATCA')) or position to set the position (*e.g.*me.position('wgs84','30deg','40deg','10m')). Although this information is available from the MeasurementSet, what location is ambiguous in some cases *e.g.* VLBI.

padding When gridding and transforming, the array may be padded by this factor in the image plane. This reduces aliasing, especially in wide-field cleaning.

usemodelcol if this is false it tells imager to create and use the model visibility on the fly and in memory as far as possible...otherwise if it is True then imager will use the MODEL_DATA column to do this.

wprojplanes this parameter is is used only of `ftmachine` is set to `wproject`. This defines how many convolution functions is used in the Wprojection gridder (a -1 implies an automatic determination).

Arguments

Inputs	
ftmachine	Fourier transform machine allowed: string Default: ft
cache	Size of gridding cache in complex pixels; default use half the memory available on computer allowed: int Default: -1
tile	Size of a gridding tile in pixels (in 1 dimension) allowed: int Default: 16
gridfunction	Gridding function allowed: string Default: SF
location	Location used in phase rotations allowed: any Default: variant
padding	Padding factor in image plane (>=1.0) allowed: double Default: 1.0
freqinterp	interpolation mode in frequency;options:- nearest, linear, cubic, spline allowed: string Default: nearest
wprojplanes	No of gridding convolution functions used in wproject-ft machine (-1 means let the code decide this number) allowed: int Default: -1
epjtablename	E-Jones table name. This is used if applypointingoffsets is set to True. allowed: string Default:
applypointingoffsets	Apply pointing offset corrections during deconvolution. allowed: bool Default: false
dopbgriddingcorrections	Correct for PB gridding before prediction of visibilities. This should be True when doing deconvolution. This should be False when predicting visibilities for model sky with no primary beam attenuation in the model. allowed: bool Default: true
cfcachedirname	Directory where convolution functions are to be (or are being) cached on the disk. allowed: string Default:
rotpastep	The PA increment in degree used for on-the-fly (OTF) rotation of the A-term in A-Projection. allowed: double Default: 5.0
pastep	The PA increment in degree used to compute the PA-rotated A-term in A-Projection. allowed: double Default: 360.0
pblimit	Primary beam limit when using PBWProjection allowed: double

Returns
bool

Example

```
- im.setoptions(cache=10000000, tile=32, gridfunction='BOX',
  location=me.location('vla'))

The above example is to tell imager to use memory to fit 10000000
complex numbers and tile the image with tiles of 32 pixels on a side.
Also it tells imager to use a box function as gridding function.  The
location parameter will make imager overide the position of the
telescope to use (the default is the one it gets from the ms).


im.open('n1333.ms')
im.selectvis(fieldid=[2:6, 8:12], spwid=[1:2])
im.defineimage(nx=800, ny=800, cellx='0.5arcsec', celly='0.5arcsec', mode='velocity', nchan=
im.setoptions(ftmachine='mosaic')
im.setvp(dovp=T)
im.setoptions(ftmachine='mosaic')
im.clean(algorithm='mfclark', model='try1', niter=200)



In the above example we are making a mosaic using the fields
2,3,4,5,6,8,9,10,11,12 and we use the mosaic ftmachine. This uses the
primary beam of the telescope as the gridding function.


im.open('coma.ms')
im.selectvis(spwid=1, fieldid=1);
mydir=me.direction('J2000', '12h30m48', '12d24m0')
im.defineimage(nx=200, cellx='30arcsec', phasecenter=mydir);
im.make('outlier1');
im.defineimage(nx=1800, cellx='30arcsec');
im.setoptions(ftmachine='wproject',wprojplanes=512, padding=1.0)
im.make('main')
im.clean(algorithm='mfclark',model=['main', 'outlier1'], niter=10000, image=['coma.image',
im.done()
```

In the above example we are using the Wprojection algorithm for 3-D imaging. We are using 512 gridding functions. Sometimes if there is a memory issue (very large images and many griding functions) we suggest the use of facetting of the image with wprojection. So the example above would be something like below. Note that when using facets only the {\tt wfclark} and {\tt wfhogbom} can be used for now. Note on how an outlier field (or flanking) field is set on an interfering source outside of the field of interest.

```
im.open('coma.ms')

im.selectvis(spwid=1, fieldid=1);
mydir = me.direction('J2000', '12h30m48', '12d24m0')
im.defineimage(nx=200, ny=200, cellx='30arcsec', celly='30arcsec', phasecenter=mydir);
im.make('outlier1');
im.defineimage(nx=3000, ny=3000, cellx='30arcsec',celly='30arcsec',facets=3);
im.setoptions(ftmachine='wproject',wprojplanes=200, padding=1.2)
im.make('main')
im.clean(algorithm='wfclark',model=['main', 'outlier1'], niter=10000)
im.done()
```

imager.setscales.html

imager.setscales - Function

2.3.1 Set the scale sizes for MultiScale Clean

Description

The multiscale clean algorithm cleans an image on a number of different scales, decomposing the image into Gaussians of these scale sizes. This function allows the user to set the number of scales used (using the nscales method), or to directly control the sizes of the scales in pixels (using the uservector method). When using the nscales method, the scales are calculated using the following formula:

$$\theta_{minor} 10.0^{(i-N\_scales/2)/2.0} \tag{2.3}$$

where θ_{min} is the fitted minor axis of the clean beam. The first value is zero.

Arguments

Inputs		
scalemethod	Method by which scales are set	
	allowed:	string
	Default:	nscales
		uservector
		nscales
nscales	Number of scales	
	allowed:	int
	Default:	5
uservector	Vector of scale sizes in pixels to use, defaults should be 0,3,10	
	allowed:	doubleArray
	Default:	0.0 3.0 10.0

Returns

bool

Example

```
- im.setscales(scalemethod='nscales', nscales=6);

Here we make six scales automatically using the method described
above.
Or we could manually choose the scales in pixel numbers as follows:

- im.setscales(scalemethod='uservector', uservector=[0,3,10,30]);


Note: 0 pixel is the delta function, so if one were to select scale 0 only
it would be  equivalent to a Hogbom clean.
```

---

imager.setsmallscalebias.html

imager.setsmallscalebias - Function

2.3.1 Set bias toward smaller scales for MultiScale Clean

Description

Arguments

Inputs		
inbias	small scale bias	
	allowed:	float
	Default:	0.6

Returns
bool

Example

```
- im.setsmallscalebias(inbias=0.6);
```

imager.settaylorterms.html

imager.settaylorterms - Function

2.3.1 Set the number of Taylor series terms for Multi-Frequency Clean

Description

The multi-frequency clean algorithm cleans an image by approximating its spectra by a Taylor series expansion. This function allows the user to set the number of Taylor terms to be used. Options are 1,2,3.

Arguments

Inputs		
ntaylorterms	Number of Taylor terms	
	allowed:	int
	Default:	2
reffreq	Reference Frequency	
	allowed:	double
	Default:	0.0

Returns

bool

Example

```
- im.settaylorterms(ntaylorterms=3);
```

imager.setsdoptions.html

imager.setsdoptions - Function

2.3.1 Set some options for single dish processing

Description

Various less-often-used options for single dish processing can be set.

scale The overall scale of the single dish data is multiplied by this factor.

weight The weight given to the single dish data in the imaging is multiplied by this factor.

convsupport This parameter can be used to change the support used in gridding single dish data in imaging. If 'PB' or 'pb' is used as the 'convtype' in setoptions this parameter is ignored as the support is defined by the primary beam. The deafult of -1 mans 1 as convsupport is used for 'box' convolution function and 3 is used for 'SF' convolution function.

pointingcolumntouse This parameter is NOT to be changed under normal circumstances. This is to be used by those who know what they are doing and want to try to use different columns in the POINTING table especially if they believe their dish direction is wrong. And if any of the `_OFFSET` columns is used do not expect to be able to use a different frame in the image setup in defineimage. Possible values are `DIRECTION, TARGET, ENCODER, POINTING_OFFSET, SOURCE_OFFSET`

truncate The truncation radius as a quantity or a float value. This parameter is effective only when 'GAUSS' or 'GJINC' is used as the 'convtype' in setoptions. You can use an unit 'pixel' to specify truncation radius as pixel value. If float value is set, or quantity without unit is set, its unit will be 'pixel'.

gwidth The width of the gaussian beam as a radius of half maximum. This parameter is effective only when 'GAUSS' or 'GJINC' is used as the 'convtype' in setoptions. You can use an unit 'pixel' to specify truncation radius as pixel value. If float value is set, or quantity without unit is set, its unit will be 'pixel'. Note that, when 'GJINC' is used as the 'convtype', gwidth doesn't directry specify width of the convolution function.

jwidth The witdh of the jinc beam as a parameter c, where jinc = J_1(pi*x/c)/(pi*x/c). This parameter is effective only when 'GJINC' is used as the 'convtype' in setoptions. You can use an unit 'pixel' to specify truncation radius as pixel value. If float value is set, or quantity without unit is set, its unit will be 'pixel'. Note that jwidth doesn't directly specify width of the convolution function.

Arguments

Inputs	
scale	Scaling applied to single dish data allowed: double Default: 1.0
weight	Weights applied to single dish data allowed: double Default: 1.0
convsupport	number of pixel for convolution support allowed: int Default: -1
pointingcolumntouse	Which Pointing Table column to use to get direction allowed: string Default: DIRECTION
truncate	truncation radius (effective only for 'GAUSS' or 'GJINC') allowed: any Default: variant -1pixel
gwidth	radius of half maximum for gaussian (effective only for 'GAUSS' or 'GJINC') allowed: any Default: variant -1pixel
jwidth	c-parameter for jinc function (effective only for 'GJINC') allowed: any Default: variant -1pixel
minweight	Minimum weight level to use for weight correction and weight based masking. allowed: double Default: 0.

Returns
bool

Example

- im.setsdoptions(scale=1.0, weight=1.0, convsupport=5)

---

imager.setvp.html

imager.setvp - Function

2.3.1 Set the voltage pattern model for subsequent processing

Description

Set the voltage pattern model (and hence, the primary beam) used for a telescope. There are currently two ways to set the voltage pattern: by using the extensive list of defaults which the system knows about, or by creating a voltage pattern description with the vpmanager. The default voltage patterns include both a high and a low frequency VP for the WSRT, a VP for each observing band at the AT, several VP's for the VLA, including the appropriate beam squint for each observing band, and Gaussian for the BIMA dishes. Due to temporary limitations in the internal structure of the visibility buffer, only one telescope's voltage pattern can be applied to a particular MeasurementSet. This will be corrected shortly.

Arguments

Inputs		
dovp	Do voltage pattern (ie, primary beam) correction	
	allowed:	bool
	Default:	false
usedefaultvp	Look up the default VP for this telescope and frequency?	
	allowed:	bool
	Default:	true
vptable	If usedefaultvp is False, provide a VP Table made with vpmanager	
	allowed:	string
	Default:	
dosquint	Activate the beam squint in the VP model	
	allowed:	bool
	Default:	false
parangleinc	Parallactice angle increment for squint application	
	allowed:	any
	Default:	variant 360deg
skyposthreshold	Sky position threshold	
	allowed:	any
	Default:	variant 180deg
telescope	Which default telescope to use; if empty use the one in encoded in MS	
	allowed:	string
	Default:	
verbose	If false, suppress some messages from being sent to the logger.	
	allowed:	bool
	Default:	true

Returns
bool

Example

```
im.setvp(dovp=True, usedefaultvp=True, dosquint=False)
```

---

imager.setweightgrid.html

imager.setweightgrid - Function

2.3.1 set the requested weight grids

Description

This is a utility function when running multi imager processes in parallel on subsection of an ms/data independently One would wish to weight the dirty image before averaging or set the imaging weight density (when using unform or Brigg's style weighting) to account for all the data being used. This is **NOT** for the general user but for people who are parallelizing at the scripting level.

Arguments

Inputs		
weight	Numeric array. Required input.	
	allowed:	any
	Default:	variant
type	Type of weight requested	
	allowed:	string
	Default:	imaging

Returns

bool

Example

```
wght=im.getweightgrid('imaging')
wght2=im2.getweightgrid('imaging')
wght=wght+wght2

im.setweightgrid(weight=wght, type='imaging')
im2.setweightgrid(weight=wght, type='imaging')
```

imager.smooth.html

imager.smooth - Function

2.3.1 Calculate an image smoothed with a Gaussian beam

Description

The model images are convolved with the specified Gaussian beam. By default (normalize=T), the beam volume is normalized to unity so that the smoothing is flux preserving. The smoothing used in restoration is not normalized.

Arguments

Inputs		
model	Name of input model	
	allowed:	stringArray
	Default:	
image	Name of output smoothed images	
	allowed:	stringArray
	Default:	
usefit	Use the fitted value (rather than that specified	
	allowed:	bool
	Default:	true
bmaj	Major axis of beam	
	allowed:	any
	Default:	variant 5.arcsec
bmin	Minor axis of beam	
	allowed:	any
	Default:	variant 5.arcsec
bpa	Position angle of beam	
	allowed:	any
	Default:	variant 0deg
normalize	Normalize volume of psf to unity	
	allowed:	bool
	Default:	true
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
- im.smooth(model='3C273XC1.clean', image='3C273XC1.clean.restored',
bmaj='2.0arcsec', bmin='2.0arcsec')
```

---

imager.stop.html

imager.stop - Function

2.3.1 stop the currently executing function asap

Description

Stop the currently executing function as soon as possible. Note that it is not always possible to stop a function.

Arguments

Returns
bool

---

imager.summary.html

imager.summary - Function

2.3.1 Summarize the current state of the imager tool

Description

Writes a summary of the properties of the imager to the default logger. This includes:

- The name of the MeasurementSet (set in construction or via the open function.
- The parameters of the image (set via defineimage)
- The current beam (set by fitpsf or setbeam.
- The selection of an ms (set via selectvis)
- The general processing options (set via setoptions)

Arguments

Returns
bool

Example

```
- im.open('3C273XC1.MS');
- im.defineimage(nx=256, ny=256)
- im.summary()
```

imager.uvrange.html

imager.uvrange - Function

2.3.1 Select data within the limit of a given range

Description

Apply a uvrange so that only points within a given uvrange are selected for further usage. To be noted selectvis if used after uvrange will reset the selected range. So selectvis should be used prior to uvrange or can be used to reset it if one changes one's mind. The points are not flagged! Further point to be noted for spectral line data the uv distance is calculated using the mean of the wavelengths of the different spectral channels selected.

Arguments

Inputs		
uvmin	Minimum uv distance allowed (wavelengths)	
	allowed:	double
	Default:	0.0
uvmax	Maximum uv distance allowed (wavelengths)	
	allowed:	double
	Default:	0.0

Returns

bool

Example

```
im.weight('uniform')
im.uvrange(0, 4000.0)
```

imager.weight.html

imager.weight - Function

2.3.1 Apply additional weighting to the visibility weights

Description

Apply visibility weighting to correct for the local density of sampling in the uv plane. The imaging weights are calculated on the fly when processing the data and can be viewed by plotweights.
To correct for visibility sampling effects, natural, uniform, radial, and Briggs weighting are supported. These work as follows. Then:

natural : minimizes the noise in the dirty image. The weight of the i-th sample is set to the inverse variance:

$$w\_i = \frac{1}{\sigma\_i^2} \tag{2.4}$$

where σ_i is the noise of the i'th sample.

radial : approximately minimizes rms sidelobes for an east-west synthesis array. The weight of the i-th sample is multiplied by the radial distance from the center of the u, v plane:

$$w\_i = w\_i\sqrt{u\_i^2 + v\_i^2} \tag{2.5}$$

uniform : For Briggs and uniform weighting, we first grid the inverse variance w_i for all selected data onto a grid of size given by the argument npixels (default to nx) and u,v cell-size given by 2/fieldofview where fieldofview is the specified field of view (defaults to the image field of view). This forms the gridded weights W_k. The weight of the i-th sample is then changed:

$$w\_i = \frac{w\_i}{W\_k} \tag{2.6}$$

where W_k is the gridded weight of the relevant cell. It may be shown that this minimizes rms sidelobes over the field of view. By changing the field of view, one may suppress the sidelobes over a region different (usually smaller) than the image size.

briggs: rmode='norm' : The weights are changed:

$$w\_i = \frac{w\_i}{1 + W\_kf^2} \tag{2.7}$$

where:

$$f^2 = \frac{(5 * 10^{-R})^2}{\frac{\sum_k W\_k^2}{\sum\_i w\_i}} \tag{2.8}$$

and R is the robust parameter. The scaling of R is such that $R = 0$ gives a good tradeoff between resolution and sensitivity. R takes value between -2.0 (close to uniform weighting) to 2.0 (close to natural).

briggs: rmode='abs' : The weights are changed:

$$w\_i = \frac{w\_i}{W\_k * R^2 + 2 * \sigma_R^2} \tag{2.9}$$

where R is the robust parameter and σ_R is the noise parameter.

For more details about Briggs (aka robust) weighting, see the Briggs thesis. Note that this weighting is *not* cumulative since the imaging weights are calculated from the specified weight (function of noise; usually $1/\sigma^2$) per visibility (actually stored in the WEIGHT column).

Arguments

Inputs		
type	Type of weighting	
	allowed:	string
	Default:	natural
rmode	Mode of briggs weighting	
	allowed:	string
	Default:	norm
		abs
		none
noise	Noise used in absolute briggs weighting	
	allowed:	any
	Default:	variant 0.0Jy
robust	Parameter in briggs weighting	
	allowed:	double
	Default:	0.0
fieldofview	Field of view for uniform weighting	
	allowed:	any
	Default:	variant 0.0arcsec
npixels	Number of pixels in the u and v directions	
	allowed:	int
	Default:	0
mosaic	Individually weight the fields of a mosaic	
	allowed:	bool
	Default:	false
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
im.weight(type='briggs', rmode='norm', robust=0.5)

Applies Briggs (robust) weighting.
```

imager.mapextent.html

imager.mapextent - Function

2.3.1 Compute map extent from given set of MSs

Description

TODO: description must be filled

Arguments

Inputs		
ref	direction reference	
	allowed:	string
	Default:	J2000
movingsource	moving source name	
	allowed:	string
	Default:	
pointingcolumntouse	POINTING table column to be used for computation	
	allowed:	string
	Default:	DIRECTION

Returns

record

Example

```
TODO: example must be filled.
```

vpmanager-Tool.html

2.3.2 vpmanager - Tool

Tool for specifying voltage patterns and primary beams
Requires:

Synopsis

Description

The vpmanager tool serves to set up a list of primary beams or voltage patterns (antenna responses) and then select in detail which of them is used for which observatory. The distinction of several antenna types for a given observatory (heterogeneous arrays) is supported.
Antenna responses can be selected from either internally hard-coded ones, or response-groups defined via an AntennaResponses table, or user-defined analytic primary beams.
Imaging and simulation routines pick up the selected response definitions and instantiate them.
The vpmanager can also create a table with the description of one or more voltage patterns (vp) or primary beams (pb). There is a mapping between telescope name and the vp or pb description. The vp description table can be read by imager's setvp method, which instantiates the corresponding voltage patterns from the descriptions and applies them to the images.
The vpmanager tool is the CASA Python object which constitutes the user interface to the VPManager C++ class. By default it is named "vp" in casapy.
The VPManager class is implemented as a singleton, i.e. internally there is only one instance at all times. This instance accessed via the static VPManager::Instance() method. It is permanent until casapy is exited and can be reinitialised via the VPManager::reset() method.
The vp tool connects to the single instance of VPManager. All settings the user makes with the tool, have effect immediately and are then used by all parts of CASA which access the VPManager class (i.e. eventually all imaging and simulation routines).
The VPManager instance keeps a simple database of available antenna responses, the *vplist*. This list is initialized at the startup of CASA or by calling the reset() method of the class. In the vp tool, the reset call can be triggered using

```
vp.reset()
```

In order to support heterogeneous interferometer arrays, VPManager permits the use of *antenna types* in addition to observatory or *telescope* names.

For defining a simple response which is only spatially scaled by frequency but otherwise constant, a simple call to the vp tool is sufficient, e.g.:

```
vp.setpbairy(telescope='ALMA',
             dishdiam='12.0m',
             blockagediam='0.75m',
             maxrad='1.784deg',
             reffreq='1.0GHz',
             dopb=True)
```

This will create a new entry in the vplist for an analytic Airy disk antenna response and make it the default response for telescope "ALMA". Subsequent requests to VPManager for a ALMA antenna response will get this Airy disk. If whole *response systems* are to be defined for a given telescope, the use of an *AntennaResponses table* is possible. Such a table can be set up using the vp tool method `createantresp()` and then connected to a telescope using a command like

```
vp.setpbantresptable(telescope='ALMA',
                     antresppath=casa['dirs']['data']
                     +'/alma/responses/AntennaResponses-ALMA-RT',
                     dopb=True)
```

where the value of the `antresppath` parameter indicates the path to the AntennaResponses table. Subsequent requests for ALMA antenna responses to VPManager will start a search in the indicated AntennaResponses table for responses matching given parameters. Presently supported search parameters in VPManager::getvp() and vp.getvp() are:

- antenna type
- observation time (used for versioning and for reference frame transformations)
- frequency (as a Measure, the reference frame is respected)
- observing direction (to support elevation and azimuth dependent responses)

An example of a call to vp.getvp() is

```
myrecord = vp.getvp(telescope='ALMA',
                    antennatype = 'DV',
                    obstime = '2009/07/24/10:00:00',
                    freq = 'TOPO 100GHz',
                    obsdirection = 'AZEL 30deg 60deg')
```

If the default antenna response for the given telescope is not defined via an AntennaResponses table, the observation parameters `obstime`, `freq`, and `obsdirection` are not needed and can be omitted. The parameter `antennatype` defaults to empty string. So if no antenna types are distinguished for the given telescope, the simplest call to getvp becomes

```
myrecord = vp.getvp(telescope='HATCREEK')
```

During initialization, VPManager will look for entries in the column "AntennaResponses" of the CASA "Observatories" table. If there are non-blank entries, the string found will be interpreted as the path to the default AntennaResponses table for the given telescope.
Note that the casacore AntennaResponses C++ class (which is used by VPManager to administrate the AntennaResponses tables) also supports the additional search parameters "receiver type" and "beam number". A general interface to the response file name search is available through the vp.getrespimagename() method. But presently this accesses only AntennaResponse tables which are entered as the default table in the Observatories table.
Generally, the vp tool methods provide functionality: to set up new analytic antenna responses, select which antenna responses from the vplist to use for which telescope and antenna type, access the contents of the vplist, create and access an AntennaResponses table, create a voltage pattern table.

Methods

vpmanager	Construct a vpmanager tool (note: the underlying VPManager is a singleton)
saveastable	Save the vp or pb descriptions as a table
loadfromtable	Load the vp or pb descriptions from a table (deleting all previous definitions)
summarizevps	Summarize the currently accumulated VP descriptions
setcannedpb	Select a vp/pb from our library of common pb models
setpbairy	Make an airy disk vp
setpbcospoly	Make a vp/pb from a polynomial of scaled cosines
setpbgauss	Make a Gaussian vp/pb
setpbinvpoly	Make a vp/pb as an inverse polynomial
setpbnumeric	Make a vp/pb from a user-supplied vector
setpbimage	Make a vp/pb from a user-supplied image
setpbpoly	Make a vp/pb from a polynomial
setpbantresptable	Declare a reference to an antenna responses table
reset	Reinitialize the VPManager (will erase all VPs and defaults defined on the command li
setuserdefault	Select the VP which is to be used by the imager for the given telescope and antenna ty
getuserdefault	Get the vp list number of the present default VP/PB for the given parameters (-1 = in
getanttypes	Return the list of available antenna types for the given parameters
numvps	Return the number of vps/pbs available for the given parameters
getvp	Return the default vps/pbs record for the given parameters
createantresp	Create a standard-format AntennaResponses table
getrespimagename	Get the image name for the given parameters from the given responses table

vpmanager.vpmanager.html

vpmanager.vpmanager - Function

2.3.2 Construct a vpmanager tool (note: the underlying VPManager is a singleton)

Description

The vpmanager constructor has no arguments.

Arguments

---

vpmanager.saveastable.html

vpmanager.saveastable - Function

2.3.2 Save the vp or pb descriptions as a table

Description

Save the vp or pb descriptions as a table. Each description is in a different row of the table.

Arguments

Inputs		
tablename	Name of table to save vp descriptions in	
	allowed:	string
	Default:	

Returns

bool

---

vpmanager.loadfromtable.html

vpmanager.loadfromtable - Function

2.3.2 Load the vp or pb descriptions from a table (deleting all previous definitions)

Description

Load the vp or pb descriptions from a table created, e.g., with saveastable().

Arguments

Inputs		
tablename	Name of table to load vp descriptions from	
	allowed:	string
	Default:	

Returns
bool

---

vpmanager.summarizevps.html

vpmanager.summarizevps - Function

2.3.2 Summarize the currently accumulated VP descriptions

Description

Summarize the currently accumulated VP descriptions to the logger.

Arguments

Inputs		
verbose	Print out full record? Otherwise, print summary.	
	allowed:	bool
	Default:	false

Returns

bool

---

vpmanager.setcannedpb.html

vpmanager.setcannedpb - Function

2.3.2 Select a vp/pb from our library of common pb models

Description

We have many vp/pb models ready to go for a variety of telescopes. If 'DEFAULT' isselected, the system default for that telescope and frequency is used.

Arguments

Inputs		
telescope	Which telescope in the MS will use this vp/pb?	
	allowed:	string
	Default:	VLA
othertelescope	If telescope=="OTHER", specify name here	
	allowed:	string
	Default:	
dopb	Should we apply the vp/pb to this telescope's data?	
	allowed:	bool
	Default:	true
commonpb	List of common vp/pb models: DEFAULT code figures it out	
	allowed:	string
	Default:	DEFAULT
dosquint	Enable the natural beam squint found in the common vp model	
	allowed:	bool
	Default:	false
paincrement	Increment in Parallactic Angle for asymmetric (ie, squinted) vp application	
	allowed:	any
	Default:	variant 720deg
usesymmetricbeam	Not currently used	
	allowed:	bool
	Default:	false

Returns

record

---

vpmanager.setpbairy.html

vpmanager.setpbairy - Function

2.3.2 Make an airy disk vp

Description

Information sufficient to create a portion of the Airy disk voltage pattern. The Airy disk pattern is formed by Fourier transforming a uniformly illuminated aperture and is given by

$$vp_p(i) = (areaRatio*2.0*j_1(x)/x - 2.0*j_1(x*lengthRatio)/(x*lengthRatio))/areaNorm, \tag{2.10}$$

where areaRatio is the dish area divided by the blockage area, lengthRatio is the dish diameter divided by the blockage diameter, and

$$x = \frac{i * maxrad * 7.016 * dishdiam/24.5m}{N_{samples} * 1.566 * 60}. \tag{2.11}$$

Arguments

Inputs	
telescope	Which telescope in the MS will use this vp/pb? allowed: string Default: VLA
othertelescope	If telescope=="OTHER", specify name here allowed: string Default:
dopb	Should we apply the vp/pb to this telescope's data? allowed: bool Default: true
dishdiam	Effective diameter of dish allowed: any Default: variant 25.0m
blockagediam	Effective diameter of subreflector blockage allowed: any Default: variant 2.5m
maxrad	Maximum radial extent of the vp/pb (scales with 1/freq) allowed: any Default: variant 0.8deg
reffreq	Frequency at which maxrad is specified allowed: any Default: variant 1.0GHz
squintdir	Offset (Measure) of RR beam from pointing center, azel frame (scales with 1/freq) allowed: any Default: variant
squintreffreq	Frequency at which the squint is specified allowed: any Default: variant 1.0GHz
dosquint	Enable the natural beam squint found in the common vp model allowed: bool Default: false
paincrement	Increment in Parallactic Angle for asymmetric (ie, squinted) vp application allowed: any Default: variant 720deg
usesymmetricbeam	Not currently used allowed: bool Default: false

Returns
record

vpmanager.setpbcospoly.html

vpmanager.setpbcospoly - Function

2.3.2 Make a vp/pb from a polynomial of scaled cosines

Description

A voltage pattern or primary beam of the form

$$VP(x) = \sum_i (coeff_i \cos^{2i}(scale_i x). \tag{2.12}$$

This is a generalization of the WSRT primary beam model.

Arguments

Inputs		
telescope	Which telescope in the MS will use this vp/pb?	
	allowed:	string
	Default:	VLA
othertelescope	If telescope=="OTHER", specify name here	
	allowed:	string
	Default:	
dopb	Should we apply the vp/pb to this telescope's data?	
	allowed:	bool
	Default:	true
coeff	Vector of coefficients of cosines	
	allowed:	doubleArray
	Default:	-1
scale	Vector of scale factors of cosines	
	allowed:	doubleArray
	Default:	-1
maxrad	Maximum radial extent of the vp/pb (scales with 1/freq)	
	allowed:	any
	Default:	variant 0.8deg
reffreq	Frequency at which maxrad is specified	
	allowed:	any
	Default:	variant 1.0GHz
isthispb	Do these parameters describe a PB or a VP?	
	allowed:	string
	Default:	PB
squintdir	Offset (Measure) of RR beam from pointing center, azel frame (scales with 1/freq)	
	allowed:	any
	Default:	variant
squintreffreq	Frequency at which the squint is specified	
	allowed:	any
	Default:	variant 1.0GHz
dosquint	Enable the natural beam squint found in the common vp model	
	allowed:	bool
	Default:	false
paincrement	Increment in Parallactic Angle for asymmetric (ie, squinted) vp application	
	allowed:	any
	Default:	variant 720deg
usesymmetricbeam	Not currently used	
	allowed:	bool
	Default:	false

Returns
record

---

vpmanager.setpbgauss.html

vpmanager.setpbgauss - Function

2.3.2 Make a Gaussian vp/pb

Description

Make a Gaussian primary beam given by

$$PB(x) = e^{-(x/(halfwidth*\sqrt{1/\log(2)}))}. \tag{2.13}$$

Arguments

Inputs		
telescope	Which telescope in the MS will use this vp/pb?	
	allowed:	string
	Default:	VLA
othertelescope	If telescope==”OTHER”, specify name here	
	allowed:	string
	Default:	
dopb	Should we apply the vp/pb to this telescope’s data?	
	allowed:	bool
	Default:	true
halfwidth	Half power half width of the Gaussian at the reffreq	
	allowed:	any
	Default:	variant 0.5deg
maxrad	Maximum radial extent of the vp/pb (scales with 1/freq)	
	allowed:	any
	Default:	variant 1.0deg
reffreq	Frequency at which maxrad is specified	
	allowed:	any
	Default:	variant 1.0GHz
isthispb	Do these parameters describe a PB or a VP?	
	allowed:	string
	Default:	PB
squintdir	Offset (Measure) of RR beam from pointing center, azel frame (scales with 1/freq)	
	allowed:	any
	Default:	variant
squintreffreq	Frequency at which the squint is specified	
	allowed:	any
	Default:	variant 1.0GHz
dosquint	Enable the natural beam squint found in the common vp model	
	allowed:	bool
	Default:	false
paincrement	Increment in Parallactic Angle for asymmetric (ie, squinted) vp application	
	allowed:	any
	Default:	variant 720deg
usesymmetricbeam	Not currently used	
	allowed:	bool
	Default:	false

Returns
record

vpmanager.setpbinvpoly.html

vpmanager.setpbinvpoly - Function

2.3.2 Make a vp/pb as an inverse polynomial

Description

The inverse polynomial describes the inverse of the VP or PB as a polynomial of even powers:

$$1/VP(x) = \sum_i coeff_i * x^{2i}. \tag{2.14}$$

Arguments

Inputs		
telescope	Which telescope in the MS will use this vp/pb?	
	allowed:	string
	Default:	VLA
othertelescope	If telescope==”OTHER”, specify name here	
	allowed:	string
	Default:	
dopb	Should we apply the vp/pb to this telescope’s data?	
	allowed:	bool
	Default:	true
coeff	Coefficients of even powered terms	
	allowed:	doubleArray
	Default:	-1
maxrad	Maximum radial extent of the vp/pb (scales with 1/freq)	
	allowed:	any
	Default:	variant 0.8deg
reffreq	Frequency at which maxrad is specified	
	allowed:	any
	Default:	variant 1.0GHz
isthispb	Do these parameters describe a PB or a VP?	
	allowed:	string
	Default:	PB
squintdir	Offset (Measure) of RR beam from pointing center, azel frame (scales with 1/freq)	
	allowed:	any
	Default:	variant
squintreffreq	Frequency at which the squint is specified	
	allowed:	any
	Default:	variant 1.0
dosquint	Enable the natural beam squint found in the common vp model	
	allowed:	bool
	Default:	false
paincrement	Increment in Parallactic Angle for asymmetric (ie, squinted) vp application	
	allowed:	any
	Default:	variant 720deg
usesymmetricbeam	Not currently used	
	allowed:	bool
	Default:	false

Returns
record

vpmanager.setpbnumeric.html

vpmanager.setpbnumeric - Function

2.3.2 Make a vp/pb from a user-supplied vector

Description

Supply a vector of vp/pb sample values taken on a regular grid between x=0 and x=maxrad. We perform sinc interpolation to fill in the lookup table.

Arguments

Inputs		
telescope	Which telescope in the MS will use this vp/pb?	
	allowed:	string
	Default:	VLA
othertelescope	If telescope=="OTHER", specify name here	
	allowed:	string
	Default:	
dopb	Should we apply the vp/pb to this telescope's data?	
	allowed:	bool
	Default:	true
vect	Vector of vp/pb samples uniformly spaced from 0 to maxrad	
	allowed:	doubleArray
	Default:	-1
maxrad	Maximum radial extent of the vp/pb (scales with 1/freq)	
	allowed:	any
	Default:	variant 0.8deg
reffreq	Frequency at which maxrad is specified	
	allowed:	any
	Default:	variant 1.0GHz
isthispb	Do these parameters describe a PB or a VP?	
	allowed:	string
	Default:	PB
squintdir	Offset (Measure) of RR beam from pointing center, azel frame (scales with 1/freq)	
	allowed:	any
	Default:	variant
squintreffreq	Frequency at which the squint is specified	
	allowed:	any
	Default:	variant 1.0GHz
dosquint	Enable the natural beam squint found in the common vp model	
	allowed:	bool
	Default:	false
paincrement	Increment in Parallactic Angle for asymmetric (ie, squinted) vp application	
	allowed:	any
	Default:	variant 720deg
usesymmetricbeam	Not currently used	
	allowed:	bool
	Default:	false

Returns
record

vpmanager.setpbimage.html

vpmanager.setpbimage - Function

2.3.2 Make a vp/pb from a user-supplied image

Description

Experimental: Supply an image of the E Jones elements. The format of the image is:

Shape nx by ny by 4 complex polarizations (RR, RL, LR, LL or XX, XY, YX, YY) by 1 channel.

Direction coordinate Az, El

Stokes coordinate All four "stokes" parameters must be present in the sequence RR, RL, LR, LL or XX, XY, YX, YY.

Frequency Only one channel is currently needed - frequency dependence beyond that is ignored.

If a complexiamge is specified the real and imaginary images is to be left empty.
The other option is to provide the real and imaginary part of the E-Jones as seperale `float` images On that case one or two images may be specified - the real (must be present) and imaginary parts (optional).
Note that beamsquint must be intrinsic to the images themselves. This will be accounted for correctly by regridding of the images from Az-El to Ra-Dec according to the parallactic angle.
antnames is the Vector of names for which this response pattern apply '*' is for all The name has to match exactly the name of the Antennas in the ANTENNA table of the MS with which you want to use this VPManager table or object.

Arguments

Inputs		
telescope	Which telescope in the MS will use this vp/pb?	
	allowed:	string
	Default:	VLA
othertelescope	If telescope==”OTHER”, specify name here	
	allowed:	string
	Default:	
dopb	Should we apply the vp/pb to this telescope’s data?	
	allowed:	bool
	Default:	true
realimage	Real part of vp as an image	
	allowed:	string
	Default:	
imagimage	Imaginary part of vp as an image	
	allowed:	string
	Default:	
compleximage	complex vp as an image of complex numbers; if specified realimage and imagimage are ignored	
	allowed:	string
	Default:	
antnames	antenna names for which this pattern is valid; default is all antennas	
	allowed:	stringArray
	Default:	*

Returns
record

---

vpmanager.setpbpoly.html

vpmanager.setpbpoly - Function

2.3.2 Make a vp/pb from a polynomial

Description

The VP or PB is described as a polynomial of even powers:

$$VP(x) = \sum_i coeff_i * x^{2i}. \tag{2.15}$$

Arguments

Inputs		
telescope	Which telescope in the MS will use this vp/pb?	
	allowed:	string
	Default:	VLA
othertelescope	If telescope==”OTHER”, specify name here	
	allowed:	string
	Default:	
dopb	Should we apply the vp/pb to this telescope’s data?	
	allowed:	bool
	Default:	true
coeff	Coefficients of even powered terms	
	allowed:	doubleArray
	Default:	-1
maxrad	Maximum radial extent of the vp/pb (scales with 1/freq)	
	allowed:	any
	Default:	variant 0.8deg
reffreq	Frequency at which maxrad is specified	
	allowed:	any
	Default:	variant 1.0GHz
isthispb	Do these parameters describe a PB or a VP?	
	allowed:	string
	Default:	PB
squintdir	Offset (Measure) of RR beam from pointing center, azel frame (scales with 1/freq)	
	allowed:	any
	Default:	variant
squintreffreq	Frequency at which the squint is specified	
	allowed:	any
	Default:	variant 1.0GHz
dosquint	Enable the natural beam squint found in the common vp model	
	allowed:	bool
	Default:	false
paincrement	Increment in Parallactic Angle for asymmetric (ie, squinted) vp application	
	allowed:	any
	Default:	variant 720
usesymmetricbeam	Not currently used	
	allowed:	bool
	Default:	false

Returns
record

vpmanager.setpbantresptable.html

vpmanager.setpbantresptable - Function

2.3.2 Declare a reference to an antenna responses table

Description

Declare a reference to an antenna responses table containing a set of VP/PB definitions.

Arguments

Inputs	
telescope	Which telescope in the MS will use this vp/pb? allowed: string Default:
othertelescope	If telescope=="OTHER", specify name here allowed: string Default:
dopb	Should we apply the vp/pb to this telescope's data? allowed: bool Default: true
antresppath	The path to the antenna responses table (absolute or relative to CASA data dir.) allowed: string Default:

Returns

bool

---

vpmanager.reset.html

vpmanager.reset - Function

2.3.2 Reinitialize the VPManager (will erase all VPs and defaults defined on the command line)

Description

Reinitialize the VPManager database. Erase all VPs and defaults defined on the command line.

Arguments

Inputs

Returns
bool

---

vpmanager.setuserdefault.html

vpmanager.setuserdefault - Function

2.3.2 Select the VP which is to be used by the imager for the given telescope and antenna type

Description

Selects the VP which is to be used by the imager for the given telescope and antenna type. Overwrites a previous default. Returns True if successful.

Arguments

Inputs	
vplistnum	The number of the vp as displayed by summarizevps(), or -1 for internal default, or -2 for unset allowed: int Default: -1
telescope	Which telescope in the MS will use this vp/pb? allowed: string Default:
anttype	Which antennatype will use this vp/pb? Default: "" = all allowed: string Default:

Returns
bool

---

vpmanager.getuserdefault.html

vpmanager.getuserdefault - Function

2.3.2 Get the vp list number of the present default VP/PB for the given parameters (-1 = internal PB, -2 = none)

Description

Get the vp list number of the present default VP/PB for the given parameters.

Arguments

Inputs	
telescope	Which telescope in the MS will use this vp/pb? allowed: string Default:
anttype	Which antennatype will use this vp/pb? Default: "" = all allowed: string Default:

Returns
int

---

vpmanager.getanttypes.html

vpmanager.getanttypes - Function

2.3.2 Return the list of available antenna types for the given parameters

Description

Get a list of the available antenna types.

Arguments

Inputs		
telescope	Telescope name	
	allowed:	string
	Default:	
obstime	Time of the observation (for versioning and reference frame calculations)	
	allowed:	any
	Default:	variant
freq	Frequency of the observation (may include reference frame, default: LSRK)	
	allowed:	any
	Default:	variant
obsdirection	Direction of the observation (may include reference frame, default: J2000). default: Zenith	
	allowed:	any
	Default:	variant AZEL 0deg 90deg

Returns
stringArray

vpmanager.numvps.html

vpmanager.numvps - Function

2.3.2 Return the number of vps/pbs available for the given parameters

Description

Can be used to, e.g., determine the number of antenna types. Note: if a global response is defined for the telescope, this will increase the count of available vps/pbs by 1.

Arguments

Inputs	
telescope	Telescope name allowed: string Default:
obstime	Time of the observation (for versioning and reference frame calculations) allowed: any Default: variant
freq	Frequency of the observation (may include reference frame, default: LSRK) allowed: any Default: variant
obsdirection	Direction of the observation (may include reference frame, default: J2000). default: Zenith allowed: any Default: variant AZEL 0deg 90deg

Returns
int

vpmanager.getvp.html

vpmanager.getvp - Function

2.3.2 Return the default vps/pbs record for the given parameters

Description

Record is empty if no matching vp/pb could be found.

Arguments

Inputs	
telescope	Telescope name allowed: string Default:
antennatype	The antenna type as a string, e.g. "DV" allowed: string Default:
obstime	Time of the observation (for versioning and reference frame calculations), e.g. 2011/12/12T00:00:00 allowed: any Default: variant
freq	Frequency of the observation (may include reference frame, default: LSRK) allowed: any Default: variant
obsdirection	Direction of the observation (may include reference frame, default: J2000), default: allowed: any Default: variant AZEL 0deg 90deg

Returns
record

vpmanager.createantresp.html

vpmanager.createantresp - Function

2.3.2 Create a standard-format AntennaResponses table

Description

The AntennaResponses table serves CASA to look up the location of images describing the response of observatory antennas. Three types of images are supported: "VP" - real voltage patterns, "AIF" - complex aperture illumination patterns, "EFP" - complex electric field patterns. For each image, a validity range can be defined in Azimuth, Elevation, and Frequency. Furthermore, an antenna type (for heterogeneous arrays), a receiver type (for the case of several receivers on the same antenna having overlapping frequency bands), and a beam number (for the case of multiple beams per antenna) are associated with each response image.
The images need to be stored in a single directory DIR of arbitrary name and need to have file names following the pattern

`obsname_beamnum_anttype_rectype_azmin_aznom_azmax_elmin_elnom_elmax_freqmin_freqnom_freqmax_`

where the individual name elements mean the following (none of the elements may contain the space character, but they may be empty strings if they are not numerical values):

obsname - name of the observatory as in the Observatories table, e.g. "ALMA"

beamnum - the numerical beam number (integer) for the case of multiple beams, e.g. 0

anttype - name of the antenna type, e.g. "DV"

rectype - name of the receiver type, e.g. ""

azmin, aznom, azmax - numerical value (degrees) of the minimal, the nominal, and the maximal Azimuth where this response is valid, e.g. "-10.5_0._10.5"

elmin, elnom, elmax - numerical value (degrees) of the minimal, the nominal, and the maximal Elevation where this response is valid, e.g. "10._45._80."

freqmin, freqnom, freqmax - numerical value (degrees) of the minimal, the nominal, and the maximal Frequency (in units of frequnit) where this response is valid, e.g. "84._100._116."

frequnit - the unit of the previous three frequencies, e.g. "GHz"

comment - any string containing only characters permitted in file names and not empty space

functype - the type of the image as defined above ("VP", "AIF", or "EFP")

The createantresp method will then extract the parameters from all the images in DIR and create the lookup table in the same directory.

Arguments

Inputs	
imdir	Path to the directory containing the response images allowed: string Default:
starttime	Time from which onwards the response is valid, format YYYY/MM/DD/hh:mm:ss allowed: string Default:
bandnames	List containing the names of the observatory's frequency bands allowed: stringArray Default:
bandminfreq	List containing the lower edges of the observatory's frequency bands, e.g. ["80GHz","120GHz"] allowed: stringArray Default:
bandmaxfreq	List containing the upper edges of the observatory's frequency bands, e.g. ["120GHz","180GHz"] allowed: stringArray Default:

Returns

bool

---

vpmanager.getrespimagename.html

vpmanager.getrespimagename - Function

2.3.2 Get the image name for the given parameters from the given responses table

Description

Given the observatory name, the antenna type, the receiver type, the observing frequency, the observing direction, and the beam number, find the applicable response image and return its name.

Arguments

Inputs	
telescope	Which telescope is described by this response? allowed: string Default:
starttime	Time at which the response has to be valid, format YYYY/MM/DD/hh:mm:ss allowed: string Default:
frequency	The frequency at which the response has to be valid, e.g. "100GHz" allowed: string Default:
functype	Type of the responsefunction requested, e.g. "EFP" allowed: string Default: ANY
anttype	Antenna type (observatory-dependent) allowed: string Default:
azimuth	Azimuth of the observation (at the location of the observatory, 0 is North), e.g. "5deg" allowed: string Default: 0deg
elevation	Elevation of the observation (at the location of the observatory, 0 is North), e.g. "60deg" allowed: string Default: 45deg
rectype	Receiver type (observatory-dependent) allowed: string Default:
beamnumber	Beam number (for the case of multiple beams per receiver) allowed: int Default: 0

Returns
string

---

simulator-Module.html

2.4 simulator - Module

Module for simulation of telescope data

Description

simulator provides a unified interface for simulation of telescope processing. It can create a MeasurementSet from scratch or read in an existing MeasurementSet. It can predict synthesis data onto the (u,v) coordinates or single dish data onto (ra,dec) points, and it can corrupt this data through Gaussian errors or through specified errors reading in (anti-) calibration tables.

In the observing phase, simulator tries to act like a (simple) telescope. You first open the name of the MeasurementSet that you wish to construct. Next you use the various set* methods to setup the observing (sources, spectral windows, etc). Each such setup should be given a unique name that will be used in the next step. Then you call the observe method for each observing scan you wish to make. Here you specify the source name, spectral window name, and observing times. After this, you have a MeasurementSet that is complete but empty. In the next phase, you fill the MeasurementSet with data from a model and then corrupt the measurements (if desired). To fill it in with a model, use the predict method. Finally, to apply errors, first set up the various effects using the relevant set* methods and then call corrupt.

Some important points:

- One call to *observe* generates one scan (all rows have the same SCAN_NUMBER).
- The start and stop times specified to *observe* need not be contiguous and so one can siulate antenna drive times.
- Currently there is no facility for patterns of observing, such as mosaicing, since it is easy to do this via sequences of calls of observe.

The following columns of the MeasurementSet are particularly important:

- **DATA** The original observed visibilities are in a column called DATA. These are normally not altered by any processing in CASA. **simulator** does write this column when it creates observations.
- **CORRECTED_DATA** During a calibration process, the visibilities may be corrected for calibration effects. These corrected visibilities are stored in CORRECTED_DATA which is created upon demand.
- **MODEL_DATA** During various phases of processing, the visibilities as predicted from some model are required. These model visibilities are stored in the column MODEL_DATA. The 'ft' task can be used to calculate the model visibilitiy for a model image.

The available tool in this module is:

- Simulator - tool for simulation

simulator-Tool.html

2.4.1 simulator - Tool

Tool for simulation
Requires:

Synopsis

Description

`simulator` provides a unified interface for simulation of telescope processing. It can create a MeasurementSet from scratch or read in an existing MeasurementSet, it can predict synthesis data onto the (u,v) coordinates or single dish data onto (ra,dec) points, and it can corrupt this data through Gaussian errors or through specific errors residing in (anti-) calibration tables. In the observing phase, `simulator` tries to act like a (simple) telescope. You first make a `simulator` tool, with the name of the MeasurementSet that you wish to construct. Next you use the various `set*` methods to set up the observing (sources, spectral windows, *etc.*). Each such setup should be given a unique name that will be used in the next step. Then you call the `observe` method for each observing scan you wish to make. Here you specify the source name, spectral windoow name, and observing times. After this, you have a MeasurementSet that is complete but empty. In the next phase, you fill the MeasurementSet with data from a model and then corrupt the measurements (if desired). To fill it in with a model, use the predict method. Finally, to apply errors, first set up the various effects using the relevant `set*` methods, and then call corrupt.
Some important points (mostly for the *cognoscenti*):

- One call to `observe` generates one scan (all rows have the same SCAN_NUMBER).
- The start and stop times specified to `observe` need not be contiguous and so one can simulate antenna drive times.
- Currently there is no facility for patterns of observing, such as mosaicing, since it is easy to do this via sequences of calls of `observe`.
- The heavy duty columns (DATA, FLAG, IMAGING_WEIGHT, *etc.* are tiled. New tiles are generated for each scan. Thus the TSM files will not get very large.

`simulator` changes some columns to the MeasurementSet to store results of processing. The following columns in the MS are particularly important:

DATA The original observed visibilities are in a column called DATA. These are normally not altered by any processing in CASA. However, this simulation program does overwrite these values.

CORRECTED_DATA During a calibration process, as carried out by *e.g.* calibrater, the visibilities may be corrected for calibration effects. This corrected visibilities are stored in a column CORRECTED_DATA which is created on demand.

MODEL_DATA During various phases of processing, the visibilities as predicted from some model are required. These model visibilities are stored in a column MODEL_DATA. The ft function of the imager tool should be used to calculate the model visibility for a model image or componentmodels.

Standard tools such as the table module and the ms can be used to access and possibly change these (and all other) columns.
simulator is a tool that performs simulation of synthesis data, including (optionally) creation of a MeasurementSet, prediction of model data, and corruption by various physical effects.

Methods

simulator	Construct a simulator tool
open	Construct a simulator tool and creating a new MeasurementSet
openfromms	Construct a simulator tool using an already existing MS
close	Close the newsimulator tool
done	Close the newsimulator tool
name	Provide the name of the attached MeasurementSet
summary	Summarize the current state
type	Return the type of this tool
settimes	Set integration time, *etc.*
observe	Observe a given configuration
observemany	Observe a given configuration
setlimits	Set limits for observing
setauto	Set autocorrelation weight
setconfig	Set the antenna configuration
setknownconfig	Set the antenna configuration to a known array
setfeed	Set the feed parameters
setfield	Set one or more observed fields
setmosaicfield	Set observed mosaic fields
setspwindow	Set one or more spectral windows
setdata	Set the data parameters selection for subsequent processing
predict	Predict astronomical data from an image
setoptions	Set various processing options
setvp	Set the voltage pattern model for subsequent processing

corrupt	Corrupt the data with visibility errors
reset	Reset the corruption terms
setbandpass	Set the bandpasses
setapply	Arrange for corruption by existing cal tables
setgain	Set the gains
settrop	Set tropospheric gain corruptions
setpointingerror	Set the Pointing error
setleakage	Set the polarization leakage
oldsetnoise	Set the noise level fixed sigma (mode=simplenoise) or Brown's equation (mode=calculate
setnoise	Set the noise level fixed sigma (mode=simplenoise) or Brown's equation using the ATM n
setpa	Corrupt phase by the parallactic angle
setseed	Set the seed for the random number generator

simulator.simulator.html

simulator.simulator - Function

2.4.1 Construct a simulator tool

Description

Create a `simulator` tool.

Arguments

Returns
simulatorobject

Example

```
# create a simulator tool
mysim = simulator();
```

---

simulator.open.html

simulator.open - Function

2.4.1 Construct a simulator tool and creating a new MeasurementSet

Description

This is used to construct `simulator` tools. A simulator tool can either be instantiated from an existing MeasurementSet, predicting and/or corrupting data on the given coordinates, or it can be used to create a fundamentally new MeasurementSet from descriptions of the array configuration and the observational parameters. This is useful for making a simulator tool which will make a MeasurementSet from scratch. In order to do this, you must also run `setconfig`, `setfield`, `setspwindow`, `setfeed`, and `settimes`. Creating the actual MS is performed by `observe`. Data can be `predict`-ed and then `corrupted`-ed. In this example, we read in the antenna coordinates from an ASCII file:

Arguments

Inputs		
ms	MeasurementSet to be created	
	allowed:	string
	Default:	

Returns

bool

Example

```
tabname = 'VLAC.LOCAL.TAB'
asciifile = 'VLAC.LOCAL.STN'
mytab=table.create()
mytab.fromascii(tabname, asciifile);
xx=[]; yy:=[]; zz:=[]; diam:=[];
xx = mytab.getcol('X');
```

```
    yy = mytab.getcol('Y');
    zz = mytab.getcol('Z');
    diam = mytab.getcol('DIAM');
    #
    sm.open('NEW1.ms')
    # do configuration
    posvla = me.observatory('vla');  #  me.observatory('ALMA') also works!
    sm.setconfig(telescopename='VLA', x=xx, y=yy, z=zz, dishdiameter=diam,
         mount='alt-az', antname='VLA',
                 coordsystem='local', referencelocation=posvla);

    # Initialize the spectral windows
    sm.setspwindow(spwname='CBand', freq='5GHz',
   deltafreq='50MHz',
   freqresolution='50MHz',
   nchannels=1,
   stokes='RR RL LR LL');
    sm.setspwindow(spwname='LBand', freq='1.420GHz',
   deltafreq='3.2MHz',
   freqresolution='3.2MHz',
   nchannels=32,
   stokes='RR LL');

    # Initialize the source and calibrater
    sm.setfield(sourcename='My cal',
sourcedirection=['J2000','00h0m0.0','+45.0.0.000'],
calcode='A');
    sm.setfield(sourcename='My source',
sourcedirection=['J2000','01h0m0.0','+47.0.0.000']);

    sm.setlimits(shadowlimit=0.001, elevationlimit='8.0deg');
    sm.setauto(autocorrwt=0.0);

    sm.settimes(integrationtime='10s', usehourangle=F,
                referencetime=me.epoch('utc', 'today'));

    sm.observe('My cal', 'LBand', starttime='0s', stoptime='300s');
    sm.observe('My source', 'LBand', starttime='310s', stoptime='720s');
    sm.observe('My cal', 'CBand', starttime='720s', stoptime='1020s');
    sm.observe('My source', 'CBand', starttime='1030s', stoptime='1500s');

    sm.setdata(spwid=1, fieldid=1);
    sm.predict(imagename='M31.MOD');
    sm.setdata(spwid=2, fieldid=2);
    sm.predict(imagename='BigLBand.MOD');
    sm.close();
```

simulator.openfromms.html

simulator.openfromms - Function

2.4.1 Construct a simulator tool using an already existing MS

Description

This is used to construct `simulator` tools operating on an existing MS. Data can be predicted and/or corrupted on the MS's given coordinates.

Arguments

Inputs		
ms	MeasurementSet to be processed	
	allowed:	string
	Default:	'MS'

Returns

bool

Example

```
sm.openfromms('3C273XC1.MS');
sm.predict('3C273XC1.imagename');
sm.setnoise(simplenoise='10mJy');
sm.setgain(interval='100s', amplitude=0.01);
sm.corrupt();
sm.close();
```

simulator.close.html

simulator.close - Function

2.4.1 Close the newsimulator tool

Description

This is used to close `newsimulator` tools. Note that the data is written to disk. This is a synonym for done.

Arguments

Returns
bool

---

simulator.done.html

simulator.done - Function

2.4.1 Close the newsimulator tool

Description

This is used to close and `newsimulator` tools. Note that the data is written to disk. This is a synonym for close.

Arguments

Returns
bool

---

simulator.name.html

simulator.name - Function

2.4.1 Provide the name of the attached MeasurementSet

Description

Returns the name of the attached MeasurementSet.

Arguments

Returns
string

---

simulator.summary.html

simulator.summary - Function

2.4.1 Summarize the current state

Description

Writes a summary of the properties of the simulator to the default logger.

Arguments

Returns
bool

---

simulator.type.html

simulator.type - Function

2.4.1 Return the type of this tool

Description

This function returns the string 'simulator'. It is used so that in a script, you can make sure this variable is a simulator `tool`.

Arguments

Returns
string

---

simulator.settimes.html

simulator.settimes - Function

2.4.1 Set integration time, *etc.*

Description

The start and stop times are referenced to `referencetime`. Use either starttime/stoptime or startha/stopha. If the hour angles are specified, then the start and stop times are calculated such that the start time is later than the reference time, but less than one day later. The hour angles refer to the first source observed.

Arguments

Inputs		
integrationtime	Integration time	
	allowed:	any
	Default:	variant 10s
usehourangle	Use starttime/stoptime as hour angles - else they are referenced to referencetime	
	allowed:	bool
	Default:	true
referencetime	Reference time for starttime and stoptime. Epoch Measure . E.g me.epoch('UTC', '50000.0d')	
	allowed:	any
	Default:	variant 50000.0d epoch measure

Returns
bool

---

simulator.observe.html

simulator.observe - Function

2.4.1 Observe a given configuration

Description

Observe a given source with a given spectral window for the specified times, including start, stop, integration, and gap times. The start and stop times are referenced to `referencetime`. Use either starttime/stoptime or startha/stopha. If the hour angles are specified, then the start and stop times are calculated such that the start time is later than the reference time, but less than one day later. The hour angles refer to the first source observed.

Arguments

Inputs		
sourcename	Name of source or field (must be specified)	
	allowed:	string
	Default:	None
spwname	Unique user-supplied name for this spectral window	
	allowed:	string
	Default:	None
starttime	Start time referenced to referenceepoch	
	allowed:	any
	Default:	variant 0s
stoptime	Stop time referenced to referenceepoch	
	allowed:	any
	Default:	variant 3600s
add_observation	Add a new line to the OBSERVATION subtable for this call	
	allowed:	bool
	Default:	false
state_sig	a new line will be added to STATE if the following don't match	
	allowed:	bool
	Default:	true
state_ref		
	allowed:	bool
	Default:	false
state_cal		
	allowed:	double
	Default:	0.0
state_load		
	allowed:	double
	Default:	0.0
state_sub_scan		
	allowed:	int
	Default:	0
state_obs_mode		
	allowed:	string
	Default:	OBSERVE_TARGET.ON_SOURCE
observer		
	allowed:	string
	Default:	CASA simulator
project		
	allowed:	string
	Default:	CASA simulation

Returns
bool

simulator.observemany.html

simulator.observemany - Function

2.4.1 Observe a given configuration

Description

Observe given sources with a given spectral window for the specified times, including start, stop, integration, and gap times. The start and stop times are referenced to `referencetime`. Use either starttime/stoptime or startha/stopha. If the hour angles are specified, then the start and stop times are calculated such that the start time is later than the reference time, but less than one day later. The hour angles refer to the first source observed.

Arguments

Inputs		
sourcenames	Name of sources	
	allowed:	stringArray
	Default:	None
spwname	Unique user-supplied name for this spectral window	
	allowed:	string
	Default:	None
starttimes	Start times referenced to referenceepoch	
	allowed:	stringArray
	Default:	0s
stoptimes	Stop time referenced to referenceepoch	
	allowed:	stringArray
	Default:	3600s
directions		
	allowed:	stringArray
	Default:	
add_observation	Add a new line to the OBSERVATION subtable for this call	
	allowed:	bool
	Default:	false
state_sig	a new line will be added to STATE if the following don't match	
	allowed:	bool
	Default:	true
state_ref		
	allowed:	bool
	Default:	false
state_cal		
	allowed:	double
	Default:	0.0
state_load		
	allowed:	double
	Default:	0.0
state_sub_scan		
	allowed:	int
	Default:	0
state_obs_mode		
	allowed:	string
	Default:	OBSERVE_TARGET#ON_SOURCE
observer		
	allowed:	string
	Default:	CASA simulator
project		
	allowed:	string
	Default:	CASA simulation

Returns
bool

---

simulator.setlimits.html

simulator.setlimits - Function

2.4.1 Set limits for observing

Description

Data are flagged for two conditions:

Below elevation limit If either of the antennas point below the specified elevation limit then the data are flagged. The elevation is calculated correctly for antennas at different locations (such as occurs in VLBI).

Shadowing If one antenna shadows another such that the fractional (geometric) blockage is greater than the specified limit then the data are flagged. No correction for blockage is made for shadowed but non-flagged points.

Arguments

Inputs		
shadowlimit	Maximum fraction of geometrically shadowed area before flagging occurs	
	allowed:	double
	Default:	1e-6
		1e-6
elevationlimit	Minimum elevation angle before flagging occurs	
	allowed:	any
	Default:	variant 10deg

Returns

bool

simulator.setauto.html

simulator.setauto - Function

2.4.1 Set autocorrelation weight

Description

Arguments

Inputs		
autocorrwt	Weight to assign autocorrelations (0=none)	
	allowed:	double
	Default:	0.0
		0.0

Returns
bool

---

simulator.setconfig.html

simulator.setconfig - Function

2.4.1 Set the antenna configuration

Description

Set the positions of the antennas. Note that the name of the telescope will control which voltage pattern is applied to the data.

Arguments

Inputs	
telescopename	Name of the telescope we are simulating (determines VP) allowed: string Default: VLA 'VLA'
x	Vector of x values of all antennas [currently m] allowed: doubleArray Default: 0 []
y	Vector of y values of all antennas [currently m] allowed: doubleArray Default: 0 []
z	Vector of z values of all antennas [currently m] allowed: doubleArray Default: 0 []
dishdiameter	Vector of diameters of all antennas [currently m] allowed: doubleArray Default: 0 []
offset	Vector of offset of all antennas [currently m] allowed: doubleArray Default: 0 []
mount	Vector of mount types of all antennas (recognized mounts are 'ALT-AZ', 'EQUATORIAL', 'X-Y', 'ORBITING', 'BIZARRE' allowed: stringArray Default: ALT-AZ []
antname	Vector of names of all antennas allowed: stringArray Default: A []
padname	Vector of names of pads or stations allowed: stringArray Default: P []
coordsystem	Coordinate system of antenna positions [x,y,z], possibilities are 'global', 'local' , 'longlat' allowed: string Default: global 'global'
referencelocation	Reference location [required for local coords] Position Measure of Coordinates of array location. E.g me.position('ITRF', '30.5deg', -20.2deg', 6000km') or me.observatory('ALMA') allowed: any Default: variant ALMA position measure

Returns
bool

Example

```
diam := [25, 25, 25, 25, 25]
xx := [50, 100, 150, 200, 250]
yy := [2, -5, -20, -50, -100]
zz := [-0.5, -1.0, -1.5, -2.0, -2.5]
posvla := dm.observatory('vla');
sm.setconfig(telescopename='VLA', x=xx, y=yy, z=zz, dishdiameter=diam,
             mount='alt-az', antname='VLA',
             coordsystem='local', referencelocation=posvla);
```

---

simulator.setknownconfig.html

simulator.setknownconfig - Function

2.4.1 Set the antenna configuration to a known array

Description

Sets the configuration to a known array such as VLAA, VLBA, EVN or ATCA6.0A. The arrays are those known to simhelper. All the information needed by setconfig is filled in.

Arguments

Inputs		
arrayname	Name of the telescope configurationwe are simulating	
	allowed:	string
	Default:	VLA
		'VLA'

Returns

bool

Example

```
sm.setknownconfig('ATCA6.0A');
```

---

simulator.setfeed.html

simulator.setfeed - Function

2.4.1 Set the feed parameters

Description

The goal is to let the feed parameters be specified for each antenna and each spectral window. At this moment, you only have the choice between 'perfect R L' and 'perfect X Y' (i.e., you cannot invent your own corrupted feeds yet). Doesn't need to be run if you want perfect R and L feeds.

Arguments

Inputs	
mode	Mode for specifying feed parameters (currently, perfect only) allowed: string Default:
x	Some very secretive feed array parameter x allowed: doubleArray Default: 0
y	Some more very secretive feed array parameter y allowed: doubleArray Default: 0
pol	Guess its the polarization of feed arrays... your guess is as good as mine....if you know better let us know please ! allowed: stringArray Default: R

Returns
bool

---

simulator.setfield.html

simulator.setfield - Function

2.4.1 Set one or more observed fields

Description

Set one or more observed fields, including name, coordinates, calibration code. Can be invoked multiple times for a complex observation. Must be invoked at least once before `observe`.
If the distance to the object is set then the phase term includes a curvature for the near-field effect at the center of the image.

Arguments

Inputs		
sourcename	Name of source or field (must be specified)	
	allowed:	string
	Default:	SOURCE
		'unknown'
sourcedirection	Direction Measure of Coordinates of source to be observed. E.g me.direction('J2000', '30.5deg','-20.2deg').	
	allowed:	any
	Default:	variant
calcode	Calibration code	
	allowed:	string
	Default:	'OBJ'
distance	Distance to the object	
	allowed:	any
	Default:	variant 0m

Returns
bool

Example

```
sm.setconfig(telescopename='VLA', x=xx, y=yy, z=zz, dishdiameter=diam,
             mount='alt-az', antname='VLA',
             coordsystem='local', referencelocation=dm.observatory('vla'));

sm.setspwindow(spwname='XBAND', freq='8GHz', deltafreq='50MHz',
               freqresolution='50MHz', nchannels=1, stokes='RR
               LL');
dir0 = me.direction('B1950',  '16h00m0.0', '50d0m0.000')
sm.setfield(sourcename='SIMU1', sourcedirection=dir0);
sm.observe('SIMU1', 'XBAND', integrationtime='10s', usehourangle=T,
           starttime='0s', stoptime='3600s',
referencetime=reftime);
```

---

simulator.setmosaicfield.html

simulator.setmosaicfield - Function

2.4.1 Set observed mosaic fields

Description

Set mosaic fields by internally invoking `setfield` multiple times. Currently only handle a rectangular mosaicing pattern. Either setfield or setmosaicfield must be invoked at least once before `observe`.
If the distance to the object is set then the phase term includes a curvature for the near-field effect at the center of the image.

Arguments

Inputs		
sourcename	Name of source or field (must be specified).	
	allowed:	string
	Default:	SOURCE
		'unknown'
calcode	Calibration code	
	allowed:	string
	Default:	"
fieldcenter	Coordinates of mosaic field center	
	allowed:	any
	Default:	variant
		MDirection
xmosp	Number of mosaic pointing in horizontal direction	
	allowed:	int
	Default:	1
ymosp	Number of mosaic pointing in vertical direction	
	allowed:	int
	Default:	1
mosspacing	Spacing between mosaic pointings	
	allowed:	any
	Default:	variant 1arcsec
distance	Distance to the object	
	allowed:	any
	Default:	variant 0m

Returns

bool

Example

```
sm.setconfig(telescopename='VLA', x=xx, y=yy, z=zz, dishdiameter=diam,
             mount='alt-az', antname='VLA',
             coordsystem='local', referencelocation=dm.observatory('vla'));

sm.setspwindow(spwname='XBAND', freq='8GHz', deltafreq='50MHz',
               freqresolution='50MHz', nchannels=1, stokes='RR
               LL');
dir0 = me.direction('B1950',  '16h00m0.0', '50d0m0.000')
sm.setmosaicfield(sourcename='SIMU1', fieldcenter=dir0,
                  xmosp=2, ymosp=2, mosspacing='154.5arcsec');
sm.settimes(integrationtime='10s');
sm.observe('SIMU1_1', 'XBAND', starttime='0s', stoptime='100s');
sm.observe('SIMU1_2', 'XBAND', starttime='110s', stoptime='210s');
sm.observe('SIMU1_3', 'XBAND', starttime='220s', stoptime='320s');
sm.observe('SIMU1_4', 'XBAND', starttime='330s', stoptime='430s');
```

---

simulator.setspwindow.html

simulator.setspwindow - Function

2.4.1 Set one or more spectral windows

Description

Set one or more spectral windows for the observations, including starting frequency, number of channels, channel increment and resolution, and stokes parameters observed. Can be invoked multiple times for a complex observation. Must be invoked at least once before `observe`.

Arguments

Inputs		
spwname	Unique user-supplied name for this spectral window	
	allowed:	string
	Default:	XBAND
		'XBAND'
freq	Starting frequency	
	allowed:	any
	Default:	variant 8.0e9Hz
deltafreq	Frequency increment per channel	
	allowed:	any
	Default:	variant 50e6Hz
freqresolution	Frequency resolution per channel	
	allowed:	any
	Default:	variant 50.e6Hz
refcode	Spectral reference code e.g. LSRK, TOPO, BARY	
	allowed:	string
	Default:	TOPO
nchannels	Number of channels	
	allowed:	int
	Default:	1
stokes	Stokes types to simulate	
	allowed:	string
	Default:	RR LL
		'RR LL'

Returns

bool

Example

```
To simulate a two spectral window (or two IF's in VLA jargon) data
set, use setpwid as follows (here we are simulating 16 channels, 50MHz
wide channel for each spectral window)


sm.setspwindow(spwname='CBAND', freq='2GHz', deltafreq='50MHz',
                   freqresolution='50MHz', nchannels=16, stokes='RR LL');

sm.setspwindow(spwname='SBAND', freq='5GHz', deltafreq='50MHz',
                   freqresolution='50MHz', nchannels=16, stokes='RR LL');


Note that the spwname is used in {\tt observe} to determine which spectral window
is used.
```

---

simulator.setdata.html

simulator.setdata - Function

2.4.1 Set the data parameters selection for subsequent processing

Description

This setup tool function selects which data are to be used subsequently. After invocation of setdata, only the selected data are operated on.

Arguments

Inputs		
spwid	Spectral Window Ids (0 relative) to select	
	allowed:	intArray
	Default:	0
fieldid	Field Ids (0 relative) to select	
	allowed:	intArray
	Default:	0
msselect	TQL select string applied as a logical "and" with the other selections	
	allowed:	string
	Default:	String

Returns
bool

---

simulator.predict.html

simulator.predict - Function

2.4.1 Predict astronomical data from an image

Description

Predict astronomical data from an image. The (u,v) coordinates already exist, either from a MeasurementSet we have read in or by generating the MeasurementSet coordinates and empty data through `create()`. We simply predict onto these coordinates.

Arguments

Inputs		
imagename	Name of image from which to predict visibilities	
	allowed:	stringArray
	Default:	
complist	Name of component list	
	allowed:	string
	Default:	String
incremental	Add this model to the existing Data Visibilities?	
	allowed:	bool
	Default:	false

Returns

bool

---

simulator.setoptions.html

simulator.setoptions - Function

2.4.1 Set various processing options

Description

For most of these, set the options for `predict` details. See also `imager` help for more details.
To simulate single dish data, use gridft=SD and gridfunction=PB.

Arguments

Inputs		
ftmachine	Fourier transform machine. Possibilities are 'ft', 'sd'	
	allowed:	string
	Default:	ft
		'ft'
cache	Size of gridding cache in complex pixels	
	allowed:	int
	Default:	0
tile	Size of a gridding tile in pixels (in 1 dimension)	
	allowed:	int
	Default:	16
gridfunction	Gridding function. String: 'SF'—'BOX'—'PB'	
	allowed:	string
	Default:	SF
		'SF'
location	Location used in phase rotations. Position Measure of Coordinates of array location. E.g me.position('ITRF', '30.5deg', '-20.2deg', '6000km') or me.observatory('ALMA')	
	allowed:	any
	Default:	variant ALMA
		position measure
padding	Padding factor in image plane (>=1.0)	
	allowed:	double
	Default:	1.3
facets	Number of facets	
	allowed:	int
	Default:	1
maxdata	Maximum data to write to a single TSM file (MB)	
	allowed:	double
	Default:	2000.0
wprojplanes	Number of projection planes when using wproject as the ft-machine	
	allowed:	int
	Default:	1

Returns

bool

Example

```
sm.setoptions(cache=10000000, tile=32, gridfunction='BOX', me.location('vla'))
```

---

simulator.setvp.html

simulator.setvp - Function

2.4.1 Set the voltage pattern model for subsequent processing

Description

Set the voltage pattern model (and hence, the primary beam) used for a Telecope. There are currently two ways to set the voltage pattern: by using the extensive list of defaults which the system knows about, or by creating a voltage pattern description with the vpmanager. The default voltage patterns include both a high and a low frequency VP for the WSRT, a VP for each observing band at the AT, several VP's for the VLA, including the approrpiate beam squint for each observing band, and Gaussian for the BIMA dishes. If you are simulating a telescope which doesn't yet exist, you will need to supply a model voltage pattern using the vpmanager.

Arguments

Inputs		
dovp	Multiply by the voltage pattern (ie, primary beam) when simulating	
	allowed:	bool
	Default:	true
usedefaultvp	Look up the default VP for this telescope and frequency?	
	allowed:	bool
	Default:	true
vptable	If usedefaultvp is false, provide a VP Table made with vpmanager	
	allowed:	string
	Default:	Table
dosquint	Activate the beam squint in the VP model	
	allowed:	bool
	Default:	true
parangleinc	Parallactice angle increment for squint application	
	allowed:	any
	Default:	variant 360deg
skyposthreshold	Position threshold on the sky for feed arrays ??	
	allowed:	any
	Default:	variant 180deg
pblimit	Primary beam limit to use in feed arrays ?	
	allowed:	double
	Default:	1.0e-2

Returns
bool

Example

```
sm.setvp(dovp=T, usedefaultvp=F, vptable='MyAlternateVLAPBModel.TAB', dosquint=F);
```

---

simulator.corrupt.html

simulator.corrupt - Function

2.4.1 Corrupt the data with visibility errors

Description

Add errors specified by the `set` functions (such as noise, gains, polarization leakage, bandpass, etc) to the visibility data. The errors are applied to the MODEL_DATA, and written to the DATA and CORRECTED_DATA columns. Note that `corrupt` handles only visibility-plane effects, not image-plane effects such as pointing errors and voltage patterns, which get applied in `predict`. Note, the function applies errors to both cross- and auto-correlation data; The auto-correlation data are corrupted properly only for the thermalnoise set by `setnoise`.

Arguments

Returns
bool

Example

```
sm,openfrommms('3C273XC1.MS');
sm.predict('3C273XC1.FAKE.IMAGE');
sm.setnoise( mode='simplenoise', simplenoise='0.1Jy');
sm.setpa( mode='calculate');
sm.corrupt();
```

simulator.reset.html

simulator.reset - Function

2.4.1 Reset the corruption terms

Description

Reset the visibility corruption terms: this means that `corrupt` introduces no errors.

Arguments

Returns
bool

---

simulator.setbandpass.html

simulator.setbandpass - Function

2.4.1 Set the bandpasses

Description

Set the level of bandpass errors. The error distributions are normal, mean zero, with the variances as specified. (Not yet implemented).

Arguments

Inputs		
mode	Mode of operation. String: 'calculate'—'table'	
	allowed:	string
	Default:	calculate
		'calculate'
table	Name of table	
	allowed:	string
	Default:	”
interval	Coherence interval e.g. '1h'	
	allowed:	any
	Default:	variant 3600s
amplitude	Variances errors in amplitude and phase	
	allowed:	doubleArray
	Default:	0.0

Returns

bool

---

simulator.setapply.html

simulator.setapply - Function

2.4.1 Arrange for corruption by existing cal tables

Description

Arrange for corruption by existing cal tables, in a manner exactly analogous to calibrater.setapply.

Arguments

Inputs		
table	Calibration table name	
	allowed:	string
	Default:	
type	Component type	
	allowed:	string
	Default:	B
		BPOLY
		G
		GSPLINE
		D
		P
		T
		TOPAC
		GAINCURVE
t	Interpolation interval (seconds)	
	allowed:	double
	Default:	0.0
field	Select on field	
	allowed:	any
	Default:	variant
interp	Interpolation type (in time)	
	allowed:	string
	Default:	aipslin
		nearest
		linear
calwt	Calibrate weights?	
	allowed:	bool
	Default:	false
spwmap	Spectral windows to apply	
	allowed:	intArray
	Default:	-1
opacity	Array-wide zenith opacity (for type='TOPAC')	
	allowed:	double
	Default:	0.0

Returns
bool

---

simulator.setgain.html

simulator.setgain - Function

2.4.1 Set the gains

Description

Set the level of gain errors. Gain drift is implemented as fractional brownian motion with rms amplitude as specified. Interval is not currently used, but future statistical models for gain errors (e.g. simple Gaussian) will use it.

Arguments

Inputs		
mode	Mode of operation. String: 'fbm'	
	allowed:	string
	Default:	fbm
		'fbm'
table	Optional name of table to write	
	allowed:	string
	Default:	''
interval	timescale for gain variations NOT USED	
	allowed:	any
	Default:	variant 10s
amplitude	amplitude scale (RMS) for gain variations [real,imag] or scalar	
	allowed:	doubleArray
	Default:	0.01
		[]

Returns

bool

---

simulator.settrop.html

simulator.settrop - Function

2.4.1 Set tropospheric gain corruptions

Description

Set the atmosphere.

Arguments

Inputs		
mode	Mode of operation - screen or individual antennas	
	allowed:	string
	Default:	screen
		'screen'
table	Name of cal table	
	allowed:	string
	Default:	"
pwv	total precipitable water vapour in mm	
	allowed:	double
	Default:	3.0
deltapwv	RMS PWV fluctuations *as a fraction of PWV parameter*	
	allowed:	double
	Default:	0.15
beta	exponent of fractional brownian motion	
	allowed:	double
	Default:	1.1
windspeed	wind speed for screen type corruption (m/s)	
	allowed:	double
	Default:	7.

Returns

bool

simulator.setpointingerror.html

simulator.setpointingerror - Function

2.4.1 Set the Pointing error

Description

Set the pointing error from a calpointing table

Arguments

Inputs		
epjtablename	Name of a table that has E-Jones errors for Pointing	
	allowed:	string
	Default:	
applypointingoffsets	Apply pointing offsets	
	allowed:	bool
	Default:	false
dopbcorrection	apply primary beam correction	
	allowed:	bool
	Default:	false

Returns
bool

---

simulator.setleakage.html

simulator.setleakage - Function

2.4.1 Set the polarization leakage

Description

Set the level of polarization leakage between feeds. Currently, no time dependence is available.

Arguments

Inputs		
mode	Mode of operation. String: 'constant'	
	allowed:	string
	Default:	constant
		'constant'
table	Optional name of table to write	
	allowed:	string
	Default:	''
amplitude	Magnitude of pol leakage [real,imag]	
	allowed:	doubleArray
	Default:	0.01
		[]
offset	Meam of pol leakage [real,imag]	
	allowed:	doubleArray
	Default:	0.
		[]

Returns

bool

simulator.oldsetnoise.html

simulator.oldsetnoise - Function

2.4.1 Set the noise level fixed sigma (mode=simplenoise) or Brown's equation (mode=calculate) OBSOLETE VERSION

Description

Set various system parameters from which the thermal (ie, random additive) noise level will be calculated.
For mode=simplenoise, one specifies the standard deviation for the noise to be added to real and imaginary parts of the visibility.
For mode=calculate, the noise will vary with dish diameter, antenna efficiency, system temperature, opacity, sky temperature, etc. The noise will increase with the airmass if `tau` is greater than zero. The noise is calculated according to the *Brown Equation* (ie, R.L. Brown's calculation of MMA sensitivity, 3Oct95):

$$\Delta S = \frac{4\sqrt{2}[T_{rx}e^{\tau A} + T_{atm}(e^{\tau A} - \epsilon_l) + T_{cmb}]}{\epsilon_q \epsilon_a \pi D^2 \sqrt{\Delta\nu \Delta t}} \tag{2.16}$$

Arguments

Inputs		
mode	Mode of operation. String: 'simplenoise'—'calculate'	
	allowed:	string
	Default:	calculate
		'simplenoise'
		'calculate'
table	Name of noise table - not currently implemented	
	allowed:	string
	Default:	''
simplenoise	Level of noise (if mode=simplenoise)	
	allowed:	any
	Default:	variant 0.0Jy
antefficiency	antenna efficiency	
	allowed:	double
	Default:	0.8
		0.8
correfficiency	Correlation efficiency	
	allowed:	double
	Default:	0.85
		0.85
spillefficiency	Forward spillover efficiency	
	allowed:	double
	Default:	0.85
		0.85
tau	Atmospheric Opacity	
	allowed:	double
	Default:	0.1
		0.1
trx	Receiver temp (ie, all non-atmospheric Tsys contributions) [K]	
	allowed:	double
	Default:	50
		50
tatmos	(Physical, not Brightness) Temperature of atmosphere [K]	
	allowed:	double
	Default:	230.0
		230.0
tcmb	Temperature of cosmic microwave background [K]	
	allowed:	double
	Default:	2.7
		2.7

Returns
bool

simulator.setnoise.html

simulator.setnoise - Function

2.4.1 Set the noise level fixed sigma (mode=simplenoise) or Brown's equation using the ATM model for frequency-dependent atmospheric opacity (mode=tsys-atm) or Brown's equation, manually specifying the zenith opacity (constant across the band) and atmospheric temperature (mode=tsys-manual)

Description

Set various system parameters from which the thermal (ie, random additive) noise level will be calculated.
For mode=simplenoise, one specifies the standard deviation for the noise to be added to real and imaginary parts of the visibility.
For mode=tsys-atm or tsys-atm, the noise will vary with dish diameter, antenna efficiency, system temperature, opacity, sky temperature, etc. The noise will increase with the airmass if tau is greater than zero. The noise is calculated according to the Brown Equation (ie, R.L. Brown's calculation of MMA sensitivity, 3Oct95): dS = 4 sqrt(2) [T_rx e$\hat{}$tau A + T_atm (e$\hat{}$tau A - epsilon_l) + T_cmb] / [epsilon_q epsilon_a pi D$\hat{2}$ sqrtDelta nu Delta t]
For mode=tsys-atm, the sky brightness temperature is calculated using an atmospheric model created for the user-input PWV. For mode=tsys-manual, the user specifies the sky brightness temperature manually.

Arguments

Inputs		
mode	Mode of operation.	
	allowed:	string
	Default:	simplenoise
		'simplenoise'
		'tsys-atm'
		'tsys-manual'
table	Name of optional cal table to write	
	allowed:	string
	Default:	''
simplenoise	Level of noise if not calculated	
	allowed:	any
	Default:	variant 0.1Jy
pground	Ground pressure for ATM model (if tsys-atm)	
	allowed:	any
	Default:	variant 560mbar
relhum	ground relative humidity for ATM model (if tsys-atm)	
	allowed:	double
	Default:	20.0
altitude	site altitude for ATM model (if tsys-atm)	
	allowed:	any
	Default:	variant 5000m
waterheight	Height of water layer for ATM model (if tsys-atm)	
	allowed:	any
	Default:	variant 200m
pwv	Precipitable Water Vapor ATM model (if tsys-atm)	
	allowed:	any
	Default:	variant 1mm
tatmos	Temperature of atmosphere [K] (if tsys-manual)	
	allowed:	double
	Default:	250.0
tau	Zenith Atmospheric Opacity (if tsys-manual)	
	allowed:	double
	Default:	0.1
antefficiency	Antenna efficiency	
	allowed:	double
	Default:	0.8
spillefficiency	Forward spillover efficiency	
	allowed:	double
	Default:	0.85
correfficiency	Correlation efficiency	
	allowed:	double
	Default:	0.88
trx	Receiver temp (ie, all non-atmospheric Tsys contributions) [K]	
	allowed:	double
	Default:	50
tground	Temperature of ground/spill [K]	
	allowed:	double
	Default:	270.0
tcmb	Temperature of cosmic microwave background [K]	
	allowed:	double
	Default:	2.73
OTF	calculate noise on-the-fly (WARNING: only experts with high-RAM machines should use False)	
	allowed:	bool

Returns
bool

---

simulator.setpa.html

simulator.setpa - Function

2.4.1 Corrupt phase by the parallactic angle

Description

Corrupt phase by the parallactic angle

Arguments

Inputs		
mode	Mode of operation. String: 'calculate'—'table'	
	allowed:	string
	Default:	calculate
		'calculate'
table	Name of table	
	allowed:	string
	Default:	"
interval	Interval for parallactic angle application, e.g. '10s'	
	allowed:	any
	Default:	variant 10s

Returns

bool

---

simulator.setseed.html

simulator.setseed - Function

2.4.1 Set the seed for the random number generator

Description

Arguments

Inputs		
seed	Seed	
	allowed:	int
	Default:	185349251
		185349251

Returns
bool

UtilityPackage.html

Chapter 3

Package Utility

Utilities useful beyond astronomical processing misc-Module.html

3.1 misc - Module

Miscellaneous tools module

logsink-Tool.html

3.1.1 logsink - Tool

tool for logsink
Requires:

Synopsis

Description

Methods

logsink	Construct a logsink tool
origin	Set the origin of the message
processorOrigin	Set the CASA processor origin
filter	Set the filter level
filterMsg	Add messages to the filter out list
clearFilterMsgList	Clear list of messages to be filter out
post	Post a message
postLocally	Post locally
localId	Get local ID
version	version of CASA
id	Get ID
setglobal	Set this logger to be the global logger
setlogfile	Set the name of file for logger output
showconsole	Choose to send messages to the console/terminal
logfile	Returns the full path of the log file
ompNumThreadsTest	Determines the number of OpenMP threads in the current parallel region using an
ompGetNumThreads	Returns the number of OpenMP threads in the current parallel region
ompSetNumThreads	Specifies the number of OpenMP threads used by default in subsequent parallel regi

logsink.logsink.html

logsink.logsink - Function

3.1.1 Construct a logsink tool

Description

Arguments

Inputs

Returns
logsink

Returns
bool

Example

---

logsink.origin.html

logsink.origin - Function

3.1.1 Set the origin of the message

Description

Sets the origin of messages to be displayed

Arguments

Inputs	
fromwhere	The origin of a log messages allowed: string Default:

Returns

bool

---

logsink.processorOrigin.html

logsink.processorOrigin - Function

3.1.1 Set the CASA processor origin

Description

Sets the CASA processor origin which is shown at the end of each log origin

Arguments

Inputs	
fromwhere	Input CASA processor origin name
	allowed: string
	Default:

Returns

bool

---

logsink.filter.html

logsink.filter - Function

3.1.1 Set the filter level

Description

Set the filter level of logging messages to be displayed. This will determine what log messages go into the log file. The logger itself can adjust what gets displayed so you could set INFO5 and then filter in the logger everything above INFO1.

Arguments

Inputs		
level	Level of messages to display to the console/log file	
	allowed:	string
	Default:	ERROR
		WARN
		INFO
		INFO1
		INFO2
		INFO3
		INFO4
		INFO5
		DEBUG
		DEBUG1
		DEBUG2
		INFO

Returns

bool

---

logsink.filterMsg.html

logsink.filterMsg - Function

3.1.1 Add messages to the filter out list

Description

Add messages to the filter out list

Arguments

Inputs	
msgList	Array of strings identifying messages to filter out
	allowed: stringArray
	Default:

Returns

void

---

logsink.clearFilterMsgList.html

logsink.clearFilterMsgList - Function

3.1.1 Clear list of messages to be filter out

Description

Clear list of messages to be filter out

Arguments

Returns
void

---

logsink.post.html

logsink.post - Function

3.1.1 Post a message

Description

If the message passes the filter, write it (same as postLocally)

Arguments

Inputs		
message	Message to be posted	
	allowed:	string
	Default:	
priority	Priority of message to be posted	
	allowed:	string
	Default:	INFO
origin	Origin of message to be posted	
	allowed:	string
	Default:	

Returns

bool

---

logsink.postLocally.html

logsink.postLocally - Function

3.1.1 Post locally

Description

If the message passes the filter, write it

Arguments

Inputs		
message	Message to be posted	
	allowed:	string
	Default:	
priority	Priority of message to be posted	
	allowed:	string
	Default:	INFO
origin	Origin of message to be posted	
	allowed:	string
	Default:	

Returns
bool

Example

logsink.localId.html

logsink.localId - Function

3.1.1 Get local ID

Description

Returns the id for this class

Arguments

Inputs

Returns
string

Example

---

logsink.version.html

logsink.version - Function

3.1.1 version of CASA

Description

Returns the version of CASA as well as sending it to the log

Arguments

Inputs

Returns

string

Example

```
casalog.version()
```

---

logsink.id.html

logsink.id - Function

3.1.1 Get ID

Description

Returns the ID of the LogSink in use

Arguments

Inputs

Returns

string

Example

---

logsink.setglobal.html

logsink.setglobal - Function

3.1.1 Set this logger to be the global logger

Arguments

Inputs		
isglobal	Use as global logger	
	allowed:	bool
	Default:	true

Returns

bool

---

logsink.setlogfile.html

logsink.setlogfile - Function

3.1.1 Set the name of file for logger output

Arguments

Inputs		
filename	filename for logger	
	allowed:	string
	Default:	casapy.log

Returns

bool

---

logsink.showconsole.html

logsink.showconsole - Function

3.1.1 Choose to send messages to the console/terminal

Arguments

Inputs		
onconsole	All messages to the console as well as log file	
	allowed:	bool
	Default:	false

Returns

bool

---

logsink.logfile.html

logsink.logfile - Function

3.1.1 Returns the full path of the log file

Description

Returns the full path of the log file

Arguments

Inputs

Returns

string

Example

```
logfile = casalog.logfile()
```

---

logsink.ompNumThreadsTest.html

logsink.ompNumThreadsTest - Function

3.1.1 Determines the number of OpenMP threads in the current parallel region using an OpenMP reduction pragma

Arguments

Inputs

Returns
int

Example

```
omp_num_thread = casalog.ompNumThreadsTest()
```

---

logsink.ompGetNumThreads.html

logsink.ompGetNumThreads - Function

3.1.1 Returns the number of OpenMP threads in the current parallel region

Arguments

Returns
int

Example

```
omp_num_thread = casalog.ompNumThreadsTest()
```

---

logsink.ompSetNumThreads.html

logsink.ompSetNumThreads - Function

3.1.1 Specifies the number of OpenMP threads used by default in subsequent parallel regions

Arguments

Inputs		
numThreads		
	allowed:	int
	Default:	1

Returns
bool

Example

```
casalog.ompSetNumThreads(2)
```

deconvolver-Tool.html

3.1.2 deconvolver - Tool

deconvolver tool
Requires:

Synopsis

Description

deconvolver is a tool that deconvolves a known point spread function from an image. A `deconvolver` must be constructed for each dirty image and point spread function for which one wishes to do processing. Multicple copies of `deconvolver` may be made at any time (provide they are given different names).

Methods

deconvolver	Construct a deconvolver tool
open	Open a new dirty image and PSF
reopen	Reopen the dirty image and PSF
close	Close the deconvolver tool
done	Terminate the deconvolver process
summary	Summarize the current state
boxmask	Construct a mask from blc, trc
regionmask	Construct a mask image from a region
clipimage	Zero all pixels where Stokes I is below a threshold
clarkclean	Make a clean image using the Clark Clean a threshold
fullclarkclean	Make a clean image using the Clark Clean a threshold
dirtyname	Return the name of the dirty-image table
psfname	Return the name of the PSF-image table
make	Make an empty image
convolve	Convolves an image with the PSF
makegaussian	Make an image with a single gaussian component
state	Return the “state” of the tool
updatestate	[A GUI builders related function]Update the GUI to reflect the currect state
clean	Make a clean image with Hogbom or MultiScale Clean
naclean	Make a clean image with Hogbom with self masking
setscales	Set the scale sizes for MultiScale Clean
ft	Fourier transform the specified model
restore	Restore the residuals

residual	Find the residuals
smooth	smooth the image
mem	Make the mem image
makeprior	Make the mem's prior image, or make a mask
mtopen	Init : Make a series of images using a Multi-Term Clean algorithm
mtclean	Make a series of images using a Multi-Term Clean algorithm
mtrestore	Restore the Multi-Term residuals
mtcalcpowerlaw	Interpret Taylor coefficients as a power law, and compute spectral index

deconvolver.deconvolver.html

deconvolver.deconvolver - Function

3.1.2 Construct a deconvolver tool

Description

This is used to construct `deconvolver` tools associated with a dirty image and point spread function. The `deconvolver` tool may then be used to generate various types of images. Note that a new executable is started every time the constructor is called.
example mydecon=casac.deconvolver() mydecon.open('dirty.image', 'psf.image')

Arguments

Inputs		
dirtyname	Dirty image to be processed. Table name.	
	allowed:	string
	Default:	dirtyname
psfname	point spread function to be processed. Table name	
	allowed:	string
	Default:	

Returns

casadeconvolver

Example

```
deco=casac.deconvolver()
deco.open('3C273XC1.dirty', '3C273XC1.psf')
deco.clean(model='3C273XC1.clean', niter=10000, gain=0.2)
deco.close()
```

deconvolver.open.html

deconvolver.open - Function

3.1.2 Open a new dirty image and PSF

Description

Close the current images and open a new dirty image and PSF instead. The current state of `deconvolver` is retained, except for the data selection.

Arguments

Inputs		
dirty	Dirty image to be processed	
	allowed:	string
	Default:	
psf	point spread function to be processed	
	allowed:	string
	Default:	
warn	Produce warning messages if psf is not provided	
	allowed:	bool
	Default:	true

Returns

bool

---

deconvolver.reopen.html

deconvolver.reopen - Function

3.1.2 Reopen the dirty image and PSF

Description

Close and reopen the current dirty and PSF images, and make new convolvers and cleaners. The main benefit of this method is to flush the residual image and replace it with the dirty image.

Arguments

Inputs

Returns
bool

---

deconvolver.close.html

deconvolver.close - Function

3.1.2 Close the deconvolver tool

Description

This is used to close `deconvolver` tools. Note that the data is written to disk. The `deconvolver` process keeps running until a done tool function call is performed.

Arguments

Returns
bool

Example

```
dc.open('3C273XC1.dirty', '3C273XC1.psf')
dc.clean(model='3C273XC1.clean');
dc.close()
```

---

deconvolver.done.html

deconvolver.done - Function

3.1.2 Terminate the deconvolver process

Description

This is used to totally stop the `deconvolver` process. It is a good idea to conserve memory use on your machine by stopping the process once you no longer need it.

Arguments

Returns
bool

Example

```
dc.open('3C273XC1.dirty', '3C273XC1.psf')
dc.clean(model='3C273XC1.clean');
dc.close()
dc.done()
```

---

deconvolver.summary.html

deconvolver.summary - Function

3.1.2 Summarize the current state

Description

Writes a summary of the properties of the deconvolver to the default logger. This includes:

- The names of the dirty image and PSF (set in construction or via the open function.
- The current beam fit

Arguments

Returns
bool

Example

```
dc.open('3C273XC1.dirty', '3C273XC1.psf')
dc.summary()
```

---

deconvolver.boxmask.html

deconvolver.boxmask - Function

3.1.2 Construct a mask from blc, trc

Description

A mask image is an image with the same shape as the other images but with values between 0.0 and 1.0 as a pixel value. Mask images are used in deconvolver to control the region selected in a deconvolution.
In the Clark CLEAN, the mask image can usefully have any value between 0.0 and 1.0. Intermediate value discourage but do not rule out selection of clean components in that region. This is accomplished by multiplying the residual image by the mask prior to entering the minor cycle. Note that if you do use a mask for the Clark or Hogbom Clean, it must cover only a quarter of the image. boxmask does not enforce this requirement.

Arguments

Inputs		
mask	name of mask image	
	allowed:	string
	Default:	
blc	Bottom left corner	
	allowed:	intArray
	Default:	-1
trc	Top right corner	
	allowed:	intArray
	Default:	-1
fillvalue	Value to fill in	
	allowed:	any
	Default:	variant 1.0Jy
outsidevalue	outside value	
	allowed:	any
	Default:	variant 0.0Jy

Returns
bool

Example

```
dc.boxmask(mask='bigmask', blc=[56,45,1,1], trc=[87,93,4,1])
dc.clean(mask='bigmask', model='3C273XC1.clean.masked', niter=1000)
```

---

deconvolver.regionmask.html

deconvolver.regionmask - Function

3.1.2 Construct a mask image from a region

Description

A mask image is an image with the same shape as the other images but with values between 0.0 and 1.0 as a pixel value. Mask images are used in imager to control the region selected in a deconvolution.
In Clark CLEAN, the mask image can usefully have any value between 0.0 and 1.0. Intermediate value is discouraged but do not rule out selection of clean components in that region. This is accomplished by multiplying the residual image by the mask prior to entering the minor cycle. Note that if you do use a mask for the Clark or Hogbom Clean, it must cover only a quarter of the image. `regionmask` does not enforce this requirement.
The function regionmask also allows multiple regions to be used. A record of the regions can be made as in the example below.
Regions can be made in many different ways using the regionmanager functions. An example using wbox function is given below. The default regionmanager tool 'rg' can be used for cases the user want to have flexibility in manipulating regions. The `region` parameter takes a record that comes from the regionmanager output. The parameter boxes allow the user to sent in a list of 4 elements numbers representing blc's and trc's
If both the parameters, `regions` and `boxes` are used the a union is done with the two sets of region thus defined.

Arguments

Inputs		
mask	name of mask image	
	allowed:	string
	Default:	
region	Region record usually from regionmanager	
	allowed:	record
	Default:	unset
boxes	list of 4 elements lists e.g [[xblc1, yblc1, xtrc1, ytrc1], [[xblc2, yblc2, xtrc2, ytrc2]]	
	allowed:	any
	Default:	variant
value	Value to set the mask to	
	allowed:	double
	Default:	1.0

Returns
bool

Example

```
Makes a mask  then cleans using it.

dc.open(dirty.image', 'psf.image')
a=[100.0, 100.0, 200, 200.0]
b=[50, 50, 80, 80]
dc.regionmask(mask='bigmask', boxes[a,b])
dc.clean(algorithm='hogbom', mask='bigmask', model='model.clean.masked', niter=1000)

Another example using rg.wbox function:
ia.open('dirty')
cs:=ia.coordsys()
rg.setcoordinates(cs.record())
r1:=dg.wbox(blc=['173pix', '347pix'], trc=['183pix', '370pix'])
c.regionmask(mask='bigmask',region=r1)

Or using a dict of regions:

r2=rg.wbox(blc=['180pix', '344pix'], trc=['191pix', '369pix'])
```

```
r3=rg.wbox(blc=['189pix', '341pix'], trc=['204pix', '364pix'])
regs={"reg1"':r1, "reg2":r2, "reg3":r3}
rec=rg.makeunion(regs)
dc.regionmask(mask='bigmask',region=rec)

If quantities are to be used to define regions the following is a an example

dc.regionmask(mask='joetest',boxes=['15:23:32.902','+05.19.32.089','15:22:28.631','+05.28.52
```

---

deconvolver.clipimage.html

deconvolver.clipimage - Function

3.1.2 Zero all pixels where Stokes I is below a threshold

Description

All pixels in the image with Stokes I less than some threshold are set to zero. This is useful prior to self-calibration where one oftens wishes to remove negative pixels from the model. Note that if the image has polarization information, then the polarized part of a pixel is also set to zero if Stokes I is less than the threshold.

Arguments

Inputs		
clippedimage	name of clipped image	
	allowed:	string
	Default:	
inputimage	name of input image	
	allowed:	string
	Default:	
threshold	Threshold	
	allowed:	any
	Default:	variant 0.0Jy

Returns

bool

Example

```
dc.clipimage(image='clean', threshold='50mJy')
###the ft that model into an MS and do gaincal for e.g
```

deconvolver.clarkclean.html

deconvolver.clarkclean - Function

3.1.2 Make a clean image using the Clark Clean a threshold

Description

In the Clark Clean algorithm, the cleaning is split into minor and major cycles. In the minor cycles only the brightest points are cleaned, using a subset of the point spread function. In the major cycle, the points thus found are subtracted correctly by using an FFT-based convolution.

Arguments

Inputs		
niter	Number of iterations	
	allowed:	int
	Default:	1000
gain	Loop Gain for CLEANing	
	allowed:	double
	Default:	0.1
threshold	Flux level at which to stop CLEANing	
	allowed:	any
	Default:	variant 0Jy
displayprogress	Display the progress of the cleaning?	
	allowed:	bool
	Default:	false
model	Name of images	
	allowed:	string
	Default:	
mask	Name of mask images used for CLEANing	
	allowed:	string
	Default:	
histbins	Number of bins in the pixel-flux histogram	
	allowed:	int
	Default:	500
psfpatchsize	Size of PSF for minor cycle	
	allowed:	intArray
	Default:	51 51
maxextpsf	maximum external sidelobe, used to set depth of minor cycle clean	
	allowed:	double
	Default:	0.2
speedup	Cleaning speedup exponent	
	allowed:	double
	Default:	0.0
maxnumpix	Maximum number of pixels used in each minor cycle	
	allowed:	int
	Default:	10000
maxnummajcycles	Max number of major cycles; -1 = no restrictions	
	allowed:	int
	Default:	-1
maxnumminoriter	Max number of minor iterations; -1 = no restrictions	
	allowed:	int
	Default:	-1

Returns
bool

deconvolver.fullclarkclean.html

deconvolver.fullclarkclean - Function

3.1.2 Make a clean image using the Clark Clean a threshold

Description

This is similar to clarkclean except that it accepts casa standard images. I.e it has to have 4 axes with the canonical order of (direction1, direction2, stokes, spectral). This function further will clean more than the quarter of the image if the mask coverage is larger than a quarter of the image size. It is useful for CLI level parallelization, i.e use imager to make a dirty image and psf and use this function to deconvolve. If in doubt use clarkclean

Arguments

Inputs		
niter	Number of iterations	
	allowed:	int
	Default:	1000
gain	Loop Gain for CLEANing	
	allowed:	double
	Default:	0.1
threshold	Flux level at which to stop CLEANing	
	allowed:	any
	Default:	variant 0Jy
model	Name of model image that will contain the clean components	
	allowed:	string
	Default:	
mask	Name of mask image used for CLEANing	
	allowed:	string
	Default:	
cyclefactor	Factor to determine how deep to go in a Clark minor cycle	
	allowed:	double
	Default:	1.5

Returns
record

deconvolver.dirtyname.html

deconvolver.dirtyname - Function

3.1.2 Return the name of the dirty-image table

Arguments

Returns
string

---

deconvolver.psfname.html

deconvolver.psfname - Function

3.1.2 Return the name of the PSF-image table

Arguments

Returns
string

---

deconvolver.make.html

deconvolver.make - Function

3.1.2 Make an empty image

Description

Make an empty image with the properties (co-ordinate system etc.) borrowed from the dirty image.

Arguments

Inputs		
image	Name of the new image on the disk	
	allowed:	string
	Default:	
async	Run asynchronously in the background?	
	allowed:	bool
	Default:	false

Returns

bool

---

deconvolver.convolve.html

deconvolver.convolve - Function

3.1.2 Convolves an image with the PSF

Description

Convolves an image (e.g., the model image) with the PSF

Arguments

Inputs	
convolvemodel	Name of the output image on the disk to hold the result of the convolution allowed: string Default:
model	The input image to be convolved with the PSF allowed: string Default:

Returns

bool

---

deconvolver.makegaussian.html

deconvolver.makegaussian - Function

3.1.2 Make an image with a single gaussian component

Description

Make a model image with the a single gaussian. The properties of the output image (e.g. the co-ordinate system, etc.) are borrowed from the dirty image. The image is made as follows:
I(x,y)= Delta function of unit amplitude at (0,0)
Temp(x,y) = Gaussian(x,y,Amplitude, Center, Sigma, PA)
I(x,y) = Convovlution of Temp(x,y) with I(x,y).
If `normalize=T` I(x,y) = I(x,y)/(area under the gaussian).

Arguments

Inputs		
gaussianimage	Name of the output image on the disk	
	allowed:	string
	Default:	
bmaj	The major axis of the gaussian	
	allowed:	any
	Default:	variant 0rad
bmin	The minor axis of the gaussian	
	allowed:	any
	Default:	variant 0rad
bpa	The Position Angle of the gaussian	
	allowed:	any
	Default:	variant 0deg
normalize	Normalize the area under the gaussian to 1.0?	
	allowed:	bool
	Default:	true
async	Run asynchronously in the background?	
	allowed:	bool
	Default:	false

Returns

bool

deconvolver.state.html

deconvolver.state - Function

3.1.2 Return the "state" of the tool

Description

Prints the name of the Dirty Image and the PSF and the parameters of the gaussian fitted to the main lobe of the PSF (the "Clean Beam").

Arguments

Returns
bool

---

deconvolver.updatestate.html

deconvolver.updatestate - Function

3.1.2 [A GUI builders related function]Update the GUI to reflect the currect state

Description

Updates the GUI to reflect the currect state of the tool. This function is used by toolmanager. See documentation of the toolmanager for details about "methods" used to update the GUI.

Arguments

Inputs		
f	Glish variable for the GUI to be updated	
	allowed:	string
	Default:	
method	The method to be used for updating	
	allowed:	string
	Default:	DONE
		close
		INIT

Returns
bool

---

deconvolver.clean.html

deconvolver.clean - Function

3.1.2 Make a clean image with Hogbom or MultiScale Clean

Description

Makes a clean image using either the Hogbom or MultiScale algorithms. The MultiScale algorithm is the default. The clean is performed on the residual image calculated from the dirty image minus the point spread function convolved with the model currently selected. Thus if you want to restart a clean, simply set the model to the model that was previously produced by clean.
Rather than explicit CLEAN boxes, mask images are used to constrain the region that is to be deconvolved. To make mask images, use either boxmask (to define a mask via the corner locations blc and trc) or mask (to define a mask via thresholding an existing image). The default mask is the inner quarter of the image.
The CLEAN deconvolution is joint in whatever Stokes parameters are present. Thus it searchs for peaks in either I or $I + |V|$ or $I + \sqrt{Q^2 + U^2 + V^2}$, the rationale for the latter two forms being to be biased towards finding strongly polarized pixels first (these forms are also the maximum eigenvalue of the coherency matrix). The PSF is constrained to be the same in all polarizations (a feature of this implementation, not of the Hamaker-Bregman-Sault formalism).
The clean algorithms possible are:

Hogbom The classic algorithm: delta function units of emission are found iteratively by searching for the peak. Each point is subtracted from the full residual image using the shifted and scaled point spread function.

Multi-Scale As the Multi-Scale Clean algorithm is quite new, we provide entensive information on its use.

In the Multi-scale Clean, the image is cleaned simultaneously with several different beams given by the point spread function convolved with components of various shapes and sizes. The components we use in this implementation are upside-down paraboloids multiplied by first order spheroidal functions (ie, the same functions used in gridding Fourier plane data). The paraboloids are truncated at zero, and the multiplication by the spheroidal function results in a smooth shape with minimal power at long baselines. This shape is scaled to the component sizes specified in setscales. As these functions have finite extent (unlike a Gaussian), they can easily be used with mask images.

For each iteration, the scale size which is able to subtract the most flux is chosen (but with a caveat, see below). The model is then built up out of the spheroidal functions of the various scale sizes. The scale sizes are set by the setscales function, which will permit the user to specify the scale sizes explicitly, or will optionally take the number of scale sizes to clean for and calculate the scale sizes themselves from a power law.

Most images deconvolved with Multi-scale Clean will be dominated by extended structure, and the largest scale size will initially remove the most flux from the dirty image. As the algorithm reduces the residuals on the largest scale, the residuals on the smaller scales will also be reduced, even without cleaning on those size scales (a falling sea sinks all boats). However, at some point, the residual image will be dominated by features on smaller size scales. These smaller features will be both positive and negative (ie, to correct for the largest size scale being the wrong shape for the true emission features). Later in the algorithm, the magnitude of the residuals on all scales will be approximately equal. At this stage, most of the deconvolvable flux has been assimilated into the largest scale size components and detailed corrections to the large scale components must be made. At this point, the user may consider switching to a faster algorithm such as the Clark Clean.

Masking is fully available with Multi-scale Clean. No component is permitted to place any of its wings outside of the user-supplied mask. If the masking were based upon the different scale components' center positions, then the large scale components could place their wings outside the mask, but the smaller scale components would not be able to make fine scale corrections. Hence, the Multi-scale Clean uses a different mask for each different size scale internally. If the mask is too restrictive or the scales are too large, the algorithm may not be able to fit the large scales into the mask at all, and the user is warned of this condition.

Traditional Clean algorithms use a small loop gain such as 0.1 to avoid confusing emission and sidelobes when extended emission is present. However, as MultiScale Clean can image large extended structure in a single spheroidal component, a loop gain in the range 0.5 to 1.0 can be used. If the largest residual oscillates between positive and negative with iteration number, as it can for some brightness distributions which include point sources, a lower loop gain will improve the imaging and the convergence.

A mild bias favoring cleaning small scale emission has been built into the Multi-scale algorithm. To illustrate the requirement of this bias, consider the case of a bright point source with very faint extended emission. Each scale may find its optimal component to subtract at the position of the bright point source, but each successively larger component will integrate more extended flux. Hence, the largeest scale component will be removed from the residuals. If most of the flux were in the point source, then several smaller negative components must be subtracted from the largest

component, and finally the point component itself may be removed after the extended emission has been taken care of. To prevent this situation from occurring, we bias the selection of small-sized components.

Note that for all of these functions except fullmsclean, only a quarter of the image may be cleaned. If no mask is set, then the cleaned region defaults to the inner quarter. If a mask larger than a quarter of the image is set, then only the quarter starting at the bottom left corner is used. Algorithm fullmsclean will deconvolve the entire field. This is useful when performing a limited accuracy deconvolution (as needed for example in wide-field imaging) but will diverge if pushed too deep.
The clean threshold may be either absolute ('0.5Jy') or relative ('1%').

Arguments

Inputs		
algorithm	Algorithm to use	
	allowed:	string
	Default:	fullmsclean
		msclean
		hogbom
niter	Number of Iterations, set to zero for no CLEANing	
	allowed:	int
	Default:	1000
gain	Loop Gain for CLEANing, try 0.7 for msclean or fullmsclean	
	allowed:	double
	Default:	0.1
threshold	Flux level at which to stop CLEANing	
	allowed:	any
	Default:	variant 0Jy
displayprogress	Display progress	
	allowed:	bool
	Default:	false
model	Name of images	
	allowed:	string
	Default:	
mask	Name of mask images used for CLEANing	
	allowed:	string
	Default:	
async	Run asynchronously in the background?	
	allowed:	bool
	Default:	false

Returns

record

Example

```
dc.clean(image='3C273XC1.clean.image', model='3C273XC1.clean.model',
mask='3C283XC1.mask', niter=1000, gain=0.25, threshold=0.03)
```

---

deconvolver.naclean.html

deconvolver.naclean - Function

3.1.2 Make a clean image with Hogbom with self masking

Description

The clean is performed on the residual image calculated from the dirty image minus the point spread function convolved with the model currently selected. Thus if you want to restart a clean, simply set the model to the model that was previously produced by clean.

Arguments

Inputs	
niter	Number of Iterations, set to zero for no CLEANing
	allowed: int
	Default: 1000
gain	Loop Gain for CLEANing, try 0.7 for msclean or fullmsclean
	allowed: double
	Default: 0.1
threshold	Flux level at which to stop CLEANing
	allowed: any
	Default: variant 0Jy
model	Name of images
	allowed: string
	Default:
mask	Name of mask images to return region cleaned
	allowed: string
	Default:
masksupp	Number of pixels on each side of peak to set the memory mask
	allowed: int
	Default: 3
memory	this define memory function to use (none, weak, medium, strong)
	allowed: string
	Default: medium
numsigma	remember positions of peak above this number of sigmas
	allowed: double
	Default: 5.0

Returns
record

Example

```
dc.naclean(image='3C273XC1.clean.image', model='3C273XC1.clean.model',
mask='3C283XC1.mask', niter=1000, gain=0.25, threshold=0.03)
```

---

deconvolver.setscales.html

deconvolver.setscales - Function

3.1.2 Set the scale sizes for MultiScale Clean

Description

Set the scale sizes, all required PSF's and Dirty Images for MultiScale Clean will be calculated. You can either give the number of scales, in which case the the scale sizes are set via a power law, or give a vector of scale sizes in pixels.

Arguments

Inputs		
scalemethod	Method by which scales are set	
	allowed:	string
	Default:	uservector
		nscales
nscales	Number of scales	
	allowed:	int
	Default:	5
uservector	Vector of scale sizes to use	
	allowed:	doubleArray
	Default:	0.0 3.0 10.0

Returns

bool

Example

```
dc.setscales(6);
```

deconvolver.ft.html

deconvolver.ft - Function

3.1.2 Fourier transform the specified model

Description

Fourier transform the specified model to an image.

Arguments

Inputs		
model	Name of image	
	allowed:	string
	Default:	
transform	Name of transform image	
	allowed:	string
	Default:	
async	Run asynchronously in the background?	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
deco.ft(model='3C273XC1.nnls.model', transform='3C273XC1.nnls.model.ft')
```

---

deconvolver.restore.html

deconvolver.restore - Function

3.1.2 Restore the residuals

Description

Restore the residuals to a smoothed version of the model. The model images are convolved with the specified Gaussian beam and then the residual images are added. If the beam is not supplied, one will be fit to the PSF.

Arguments

Inputs		
model	Name of input model	
	allowed:	string
	Default:	
image	Name of output restored image	
	allowed:	string
	Default:	
bmaj	Major axis of beam	
	allowed:	any
	Default:	variant 0rad
bmin	Minor axis of beam	
	allowed:	any
	Default:	variant 0rad
bpa	Position angle of beam	
	allowed:	any
	Default:	variant 0deg
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
deco.restore(model='3C273XC1.clean', image='3C273XC1.clean.restored',
bmaj='2.0arcsec', bmin='2.0arcsec')
```

---

deconvolver.residual.html

deconvolver.residual - Function

3.1.2 Find the residuals

Description

Calculate the residuals corresponding to the model. componentlist.

Arguments

Inputs		
model	Names of input models	
	allowed:	string
	Default:	
image	Names of output residual images	
	allowed:	string
	Default:	
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
deco.residual(model='3C273XC1.clean', complist='3C273XC1.cl',
image='3C273XC1.clean.residual')
```

deconvolver.smooth.html

deconvolver.smooth - Function

3.1.2 smooth the image

Description

The model image is convolved with the specified Gaussian beam. By default (normalize=T), the beam volume is normalized to unity so that the smoothing is flux preserving. The smoothing used in restoration is not normalized.

Arguments

Inputs		
model	Name of input model	
	allowed:	string
	Default:	
image	Name of output smoothed image	
	allowed:	string
	Default:	
bmaj	Major axis of beam	
	allowed:	any
	Default:	variant 0rad
bmin	Minor axis of beam	
	allowed:	any
	Default:	variant 0rad
bpa	Position angle of beam	
	allowed:	any
	Default:	variant 0deg
normalize	Normalize volume of psf to unity	
	allowed:	bool
	Default:	true
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
- deco.smooth(model='3C273XC1.clean', image='3C273XC1.clean.restored',
bmaj='2.0arcsec', bmin='2.0arcsec')
```

---

deconvolver.mem.html

deconvolver.mem - Function

3.1.2 Make the mem image

Description

Makes a mem image using the Cornwell-Evans algroithm, using either maximum entropy (entropy) or maxmimum emptiness (emptiness). The maximum entropy algorithm is the default. You can restart a MEM deconvolution on an existing model image, but the alpha and beta parameters are not yet saved.
Mask images can be used to restrict where the algorithm puts flux. A prior, or bias, image can provide a priori information to the algorithm and effectively limit the support as well as a mask. The prior image can be constructed by smoothing an existing estimate for the brightness distribution and clipping. Any pixel values below 1e-6 will be clipped to this level, so zero or negative pixels will not cause problems.
Currently, only one Stokes parameter may be deconvolved at a time. Stokes I images can be deconvolved with either maximum entropy or maxmimum emptiness. Stokes Q, U, or V should be deconvolved with maxmimum emptiness, which permits negative pixel values. Joint polarization MEM deconvolution is planned for the future.
The mem entropies possible are:

entropy The smoothness of the image, relative to some prior (also called default or bias) image is maximized. The functional form of the entropy is $H = \sum Iln(I/M)$, where I is the mem image brightness and M is the prior image. As the prior image is positive definite, the entropy constrains the mem image pixels to be positive, hence only stokes I can be imaged.

emptiness The number of pixels with absolute value of the flux greater than the noise level is minimized. This treats positive and negative pixel values equally, so it is appropriate for any Stokes image.

This MEM algorithm works in the image plane (ie, is ignorant of visibility data), but performs the convolution by multiplication in the Fourier plane. Not to be confused with this usage of the term "image plane", some problems are "image plane" problems, such as a single dish performing On-The-Fly mapping. Independent noise is added at each integration as the beam sweeps over the object (ie, in the image plane). This can lead to a noise signal at non-physically large spatial frequencies. This non-physical signal can be

removed by convolving the residual image with the PSF. Also key to this problem is that the PSF is of finite extent, permitting the deconvolution of nearly the entire dirty image rather than just the inner quarter. These options are accessed by setting `imageplane` to T.

Arguments

Inputs		
entropy	entropy to use	
	allowed:	string
	Default:	emptiness
		entropy
niter	Number of Iterations, set to zero for no MEMing	
	allowed:	int
	Default:	20
sigma	Noise level to try to achieve	
	allowed:	any
	Default:	variant 0.001Jy
targetflux	Total image flux to try to achieve	
	allowed:	any
	Default:	variant 1.0Jy
constrainflux	Use targetflux as a constraint? (or starting flux)	
	allowed:	bool
	Default:	false
displayprogress	Display progress	
	allowed:	bool
	Default:	false
model	Name of input/output model image	
	allowed:	string
	Default:	
prior	Name of prior (default) image used for mem	
	allowed:	string
	Default:	
mask	Mask image restricting emission (all pixels 0 or 1)	
	allowed:	string
	Default:	
imageplane	Is this an image plane problem (like single dish)?	
	allowed:	bool
	Default:	false
async	Run asynchronously in the background?	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
deco.mem(entropy='entropy', niter=30, sigma=0.01, targetflux=10.0,
model='3C273XC1.mem.image', prior='3C283XC1.prior')
```

---

deconvolver.makeprior.html

deconvolver.makeprior - Function

3.1.2 Make the mem's prior image, or make a mask

Description

Makes a prior image for the mem function. A general way to make a prior image is to start with a low resolution image, obtained from a smaller array configuration or a lower frequency observation, from another image which has been smoothed, or from a single dish image. The low resolution image can then be doctored via clipping and regioning to make it acceptable for the mem function.
Currently, only one Stokes parameter may be used at a time.

Arguments

Inputs		
prior	output prior image	
	allowed:	string
	Default:	
templateimage	starting point for prior image	
	allowed:	string
	Default:	
lowclipfrom	Clip any pixel below this level	
	allowed:	any
	Default:	variant 0.0Jy
lowclipto	Any clipped pixel will be given this value	
	allowed:	any
	Default:	variant 0.0Jy
highclipfrom	Clip any pixel above this level	
	allowed:	any
	Default:	variant 9e20Jy
highclipto	Any clipped pixel will be given this value	
	allowed:	any
	Default:	variant 9e20Jy
blc	Bottom left hand corner for box; outside box is clipped	
	allowed:	intArray
	Default:	-1
trc	Top right hand corner for box; outside box is clipped	
	allowed:	intArray
	Default:	-1
async	Run asynchronously in the background?	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
deco.makeprior(prior='3C283XC1.prior', templateimage='3C283XC1.mem.smooth',
clipfrom='0.01Jy', clipto='0.0001Jy', blc=[100,100], trc=[150,150])
```

deconvolver.mtopen.html

deconvolver.mtopen - Function

3.1.2 Init : Make a series of images using a Multi-Term Clean algorithm

Description

Makes a series of images.
For N terms in the polynomial, supply a list of 2N-1 Hessian elements (psfs), and the scale sizes.

Arguments

Inputs		
ntaylor	Number of terms in the taylor polynomial	
	allowed:	int
	Default:	2
scalevector	Vector of scale sizes to use	
	allowed:	doubleArray
	Default:	0.0 3.0 10.0
psfs	Intpu : List of names of 2N-1 psfs. This is valid only for a Taylor-polynomial model.	
	allowed:	stringArray
	Default:	
async	Run asynchronously in the background?	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
xxx
```

deconvolver.mtclean.html

deconvolver.mtclean - Function

3.1.2 Make a series of images using a Multi-Term Clean algorithm

Description

Makes a series of images.
Supply a list of N residual images and a corresponding list of 2N-1 Hessian elements (psfs).
This way, for each field or partial-run, one can choose how many terms to operate with, changing this number depending on the signal-to-noise level.

Arguments

Inputs	
residuals	Input : List of names of N residual images
	allowed: stringArray
	Default:
models	Output : List of names of N model images
	allowed: stringArray
	Default:
niter	Number of Iterations, set to zero for no CLEANing
	allowed: int
	Default: 1000
gain	Loop Gain for CLEANing, try 0.7 for msclean or fullmsclean
	allowed: double
	Default: 0.1
threshold	Flux level at which to stop CLEANing
	allowed: any
	Default: variant 0Jy
displayprogress	Display progress
	allowed: bool
	Default: false
mask	Name of mask images used for CLEANing
	allowed: string
	Default:
async	Run asynchronously in the background?
	allowed: bool
	Default: false

Returns
record

Example

```
xxx
```

---

deconvolver.mtrestore.html

deconvolver.mtrestore - Function

3.1.2 Restore the Multi-Term residuals

Description

The model images are smoothed by the specified Gaussian beam. The principal solution is computed from the residuals. The smoothed models are added to these new residuals.
This ensures that any undeconvolved source flux has been transformed into the Taylor-coefficient basis, before being added into the smoothed Taylor-coefficient model images.
(If the beam is not supplied, one will be fit to the PSF).

Arguments

Inputs		
models	Input : Name of input model	
	allowed:	stringArray
	Default:	
residuals	Input : Name of residual image	
	allowed:	stringArray
	Default:	
images	Output : Name of output restored image	
	allowed:	stringArray
	Default:	
bmaj	Major axis of beam	
	allowed:	any
	Default:	variant 0rad
bmin	Minor axis of beam	
	allowed:	any
	Default:	variant 0rad
bpa	Position angle of beam	
	allowed:	any
	Default:	variant 0deg
async	Run asynchronously in the background	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
deco.restore(model='3C273XC1.clean', image='3C273XC1.clean.restored',
bmaj='2.0arcsec', bmin='2.0arcsec')
```

---

deconvolver.mtcalcpowerlaw.html

deconvolver.mtcalcpowerlaw - Function

3.1.2 Interpret Taylor coefficients as a power law, and compute spectral index

Description

Take ratios of restored images to compute alpha and beta

Arguments

Inputs	
images	Input : Names of input restored images
	allowed: stringArray
	Default:
residuals	Input : Names of input residuals images (for error calcs)
	allowed: stringArray
	Default:
alphaname	Output : Name of output spectral-index image
	allowed: string
	Default:
betaname	Output : Name of output spectral-curvature image
	allowed: string
	Default:
threshold	Threshold
	allowed: any
	Default: variant 0.0Jy
calcerror	Calculate an error image for spectral index
	allowed: bool
	Default: false
async	Run asynchronously in the background
	allowed: bool
	Default: false

Returns
bool

Example

```
deco.restore(model='3C273XC1.clean', image='3C273XC1.clean.restored',
bmaj='2.0arcsec', bmin='2.0arcsec')
```

---

table-Module.html

3.2 table - Module

Casapy interface to table system

Description CASA stores all data inside CASA tables which are stored on disk. A CASA table consists of an unlimited number of columns of data, with optional column keywords and optional table keywords. Columns are named and rows are numbered (starting at 0). The columns hold data, such as visibilities and uv coordinates, while the keywords hold general information such as units or revision numbers or table author or even other tables.

To make this concrete, examples of columns might be:

```
  U       V        W         TIME         ANT1   ANT2      VISIBILITY
124.011 54560.0  3477.1  43456789.0990    1      2        4.327 -0.1132
34561.0 45629.3  3900.5  43456789.0990    1      3        5.398 0.4521
....
....
....
```

and examples of keywords might be:

```
REVISION=2.01
AUTHOR="Tim Cornwell"
INSTRUMENT="VLA"
```

Everything in a CASA table (and thus all data stored in CASA) is potentially accessible and changable from casapy. The table module provides a convenient way of accessing and changing CASA tables from inside casapy. To do so one uses *tb*, the table tool, inside casapy. The table tool provides functions for:

- Opening and copying of existing tables, (using open, and copy),

- Get and put of table cells, columns and keywords,
- Selection and sorting with TaQL (using the query or calc functions),
- Get and put of table information strings,
- Browsing of tables (using the browse function),
- Printing of a summary of a table (using summary function),
- Reading or writing a table from or to an ASCII format (using fromascii, and toasciifmt)

All operations are done inside the casapy client and are not written to disk until an explicit flush command is performed.

The most typical operation on a CASA table is to open it, load a column from the table into casapy, alter it using casapy capabilities, and then write it back to the table. For this only a few commands are relevant: see the example below.

Sorting and selecting of tables is possible. The table thus produced is a *reference* table, and points back to the original table.

Example

```
tb.open('3C273XC1.MS')
tb.summary();
uvw=tb.getcol("UVW");
tb.close()
tb.open("3C273XC1.MS/SPECTRAL_WINDOW")
freq=tb.getcell("REF_FREQUENCY", 0)
tb.close()
for i in range(len(uvw)):
  for j in range(len(uvw[0])):
     uvw[i][j] = uvw[i][j]*1.420E9/freq
tb.open('3C273XC1.MS', nomodify=False)
tb.putcol("UVW", uvw);
tb.close()
```

See Also

tableplot

table-Tool.html

3.2.1 table - Tool

Access tables from casapy
Requires:

Synopsis

Description

table is the tool that contains all the functions relevant for table handling.

Methods

fromfits	Create a CASA table from a binary FITS file
fromascii	Create a CASA table from a file containing data in ASCII format
open	open an existing table
create	create a new table
flush	flush the current contents to disk
fromASDM	Create an CASA table from an ASDM table
resync	resync the table tool with table file
close	close the table tool
copy	copy a table
copyrows	copy rows from this table to another
done	end the table tool
iswritable	is the table writable?
endianformat	get the endian format used for this table
lock	acquire a lock on the table
unlock	unlock and flush the table
datachanged	has data changed in table?
haslock	has this process a lock on the table?
lockoptions	get the lock options used for this table
ismultiused	is the table in use in another process?
browse	browse a table using a graphical browser
name	return name of table on disk
createmultitable	Create a virtually concatenated table
toasciifmt	Write CASA table into an ASCII format
taql	Make a table from a TaQL command.
query	Make a table from a query
calc	TaQL expression with calc to calculate an expression on a table
selectrows	Make a table from a selection of rows

info	get the info record
putinfo	set the info record
addreadmeline	add a readme line to the info record
summary	summarize the contents of the table
colnames	return the names of the columns
rownumbers	!!!INPUT PARAMETERS IGNORED!!! return the row numbers in the (reference) tabl
setmaxcachesize	set maximum cache size for column in the table
isscalarcol	is the specified column scalar?
isvarcol	tell if column contains variable shaped arrays
coldatatype	return the column data type
colarraytype	return the column array type
ncols	return number of columns
nrows	return number of rows
addrows	add a specified number of rows
removerows	remove the specified rows
addcols	!!!REQUIRES COLUMN DESCRIPTION FUNCTIONS THAT HAVE NOT BEEN IM
renamecol	rename a column
removecols	remove one or more columns
iscelldefined	test if a specific cell contains a value
getcell	get a specific cell
getcellslice	get a slice from a specific cell
getcol	get a specific column
getvarcol	get a specific column (for variable arrays)
getcolslice	get a slice from a specific columnarray
putcell	put a specific cell
putcellslice	put a slice into a specific cell
putcol	put a specific column
putvarcol	put a specific column (for variable arrays)
putcolslice	put a slice into a specific column
getcolshapestring	get shape of arrays in a specific column
getkeyword	get value of specific table keyword
getkeywords	get values of all table keywords
getcolkeyword	get value of specific column keyword
getcolkeywords	get values of all keywords for a column
putkeyword	put a specific table keyword
putkeywords	!!!BROKEN!!! put multiple table keywords
putcolkeyword	put a specific keyword for a column
putcolkeywords	put multiple keywords for a column
removekeyword	remove a specific table keyword
removecolkeyword	remove a specific keyword for a column
getdminfo	get the info about data managers
keywordnames	get the names of all table keywords
fieldnames	get the names of fields in a table keyword
colkeywordnames	get the names of all keywords in a column
colfieldnames	get the names of fields in a keyword in a column
getdesc	get the table description

getcoldesc	get the description of a specific column
ok	Is the table tool ok?
clearlocks	Clears any table lock associated with the current process
listlocks	Lists any table lock associated with the current process
statistics	Get statistics on the selected table column
showcache	show the contents of the table cache
testincrstman	Checks consistency of an Incremental Store Manager bucket layout

table.fromfits.html

table.fromfits - Function

3.2.1 Create a CASA table from a binary FITS file

Description

Create a table from binary FITS format. This generates a CASA table from the binary FITS table in the given HDU (header unit) of the FITS file. Note that other FITS formats (*e.g.* Image FITS and UVFITS) are read by other means.
It is possible to specify the storage manager to use for the table:
`standard` is the default storage manager.
`incremental` is efficient for slowly varying data.
`memort` is for in memory use for e.g to grab given columns via getcol.

Arguments

Inputs		
tablename	Name of table to be created	
	allowed:	string
	Default:	
fitsfile	Name of FITS file to be read	
	allowed:	string
	Default:	
whichhdu	Which HDU to read (0-relative to primary HDU i.e. 1 is the smallest valid value)	
	allowed:	int
	Default:	1
storage	Storage manager to use (standard or incremental or memory)	
	allowed:	string
	Default:	standard
convention	Convention to use (sdfits or none)	
	allowed:	string
	Default:	none
nomodify	Open Read-only?	
	allowed:	bool
	Default:	true
ack	Acknowledge creations, etc	
	allowed:	bool
	Default:	true

Returns
table

---

table.fromascii.html

table.fromascii - Function

3.2.1 Create a CASA table from a file containing data in ASCII format

Description

Create a table from an ASCII file. Columnar data as well as table and column keywords may be specified.
Once the table is created from the ASCII data, it is opened in the specified mode by the table tool.
The table columns are filled from a file containing the data values separated by a separator (one line per table row). The default separator is a blank. Blanks after the separator are ignored.
If a non-blank separator is used, values can be empty. Such values default to 0, empty string, or F depending on the data type. E.g. 1,,2, has 4 values of which the 2nd and 4th are empty and default to 0. Similarly if fewer values are given than needed, the missing values get the default value.
Either the data format can be explicitly specified or it can be found automatically. The former gives more control in ambiguous situations. Both scalar and array columns can be generated from the ASCII input. The format string determines the type and optional shape.
In automatic mode (`autoheader=True`) the first line of the ASCII data is analyzed to deduce the data types. Only the types I, D, and A can be recognized. A number without decimal point or exponent is I (integer), otherwise it is D (double). Any other string is A (string). Note that a number may contain a leading sign (+ or -). The `autoshape` argument can be used to specify if the input should be stored as multiple scalars (the default) or as a single array. In the latter case one axis in the shape can be defined as variable length by giving it the value 0. It means that the actual array shape in a row is determined by the number of values in the corresponding input line.
Columns get the names `Column1`, `Column2`, etc..
For example:

1. `autoshape=[]` (which is the default) means that all values are to be stored as scalar columns.

2. `autoshape=0` means that all values in a row are to be stored as a variable length vector.

3. `autoshape=10` defines a fixed length vector. If an input line contains less than 10 values, the vector is filled with default values. If more than 10 values, the latter values are ignored.

4. `autoshape=[5,0]` defines a 2-dim array of which the 2nd axis is variable. Note that if an input line does not contain a multiple of 5 values, the array is filled with default values.

If the format of the table is explicitly specified, it has to be done either in the first two lines of the data file (named by the argument filename), or in a separate header file (named by the argument headerfile). In both forms, table keywords may also be specified before the column definitions. The column names and types can be described by two lines:

1. The first line contains the names of the columns. These names may be enclosed in quotes (either single or double).
2. The second line contains the data type and optionally the shape of each column. Valid types are:
   - S for Short data
   - I for Integer data
   - R for Real data
   - D for Double Precision data
   - X for Complex data (Real followed by Imaginary)
   - Z for Complex data (Amplitude then Phase)
   - DX for Double Precision Complex data (Real followed by Imaginary)
   - DZ for Double Precision Complex data (Amplitude then Phase)
   - A for ASCII data (a value must be enclosed in single or double quotes if it contains whitespace)
   - B for Boolean data (False are empty string, 0, or any string starting with F, f, N, or n).

If a column is an array, the shape has to be given after the data type without any whitespace. E.g. `I10` defines an integer vector of length 10. `A2,5` defines a 2-dim string array with shape [2,5]. Note that `I` is not the same as `I1` as the first one defines a scalar and the other one a vector with length 1. The last column can have one variable length axis denoted by the value 0. It "consumes" the remainder of the input line.
If the argument headerfile is set then the header information is read from that file instead of the first lines of the data file.
To give a simple example of the form where the header information is located at the top of the data file:

```
COLI    COLF    COLD        COLX          COLZ          COLS
 I       R       D           X             Z             A
1       1.1     1.11        1.12 1.13     1.14 1.15     Str1
10      11      12          13   14       15   16       ""
```

Note that a complex number consists of 2 numbers.
Also note that an empty string can be given.
Let us now give an example of a separate header file that one might use to get interferometer data into CASA:

```
U     V      W          TIME         ANT1        ANT2      DATA
R     R      R           D            I           I        X1,0
```

The data file would then look like:

```
124.011 54560.0  3477.1  43456789.0990    1       2         4.327 -0.1132
34561.0 45629.3  3900.5  43456789.0990    1       3         5.398 0.4521
```

Note that the DATA column is defined as a 2-dim array of 1 correlation and a variable number of channels, so the actual number of channels is determined by the input. In this example both rows will have 1 channel (note that a complex value contains 2 values).
Tables may have keywords in addition to the columns. The keywords are useful for holding information that is global to the entire table (such as author, revision, history, *etc,*).
The keywords in the header definitions must preceed the column descriptions. They must be enclosed between a line that starts with ".key..." and a line that starts with ".endkey..." (where ... can be anything). Between these two lines each line should contain the following as listed below. A table keywordset and column keywordsets can be specified. The latter can be specified by specifying the column name after the .keywords string.

- The keyword name, e.g., ANYKEY
- The datatype and optional shape of the keyword (cf. list of valid types above)
- The value or values for the keyword (the keyword may contain a scalar or an array of values). e.g., 3.14159 21.78945

Thus to continue the example above, one might wish to add keywords as follows:

```
.keywords
DATE        A  "97/1/16"
REVISION    D 2.01
AUTHOR      A "Tim Cornwell"
INSTRUMENT  A "VLA"
.endkeywords
.keywords TIME
UNIT A "s"
.endkeywords
U     V      W          TIME         ANT1        ANT2      DATA
R     R      R           D            I           I        X1,0
```

Similarly to the column format string, the keyword formats can also contain shape information. The only difference is that if no shape is given, a keyword can have multiple values (making it a vector).
It is possible to ignore comment lines in the header and data file by giving the `commentmarker`. It indicates that lines starting with the given marker are ignored. Note that the marker can be a regular expression (e.g. texttt' *//' tells that lines starting with // and optionally preceeded by blanks have to be ignored).
With the arguments `firstline` and `lastline` one can specify which lines have to be taken from the input file. A negative value means 1 for `firstline` or end-of-file for `lastline`. Note that if the headers and data are combined in one file, these line arguments apply to the whole file. If headers and data are in separate files, these line arguments apply to the data file only.
Also note that ignored comment lines are counted, thus are used to determine which lines are in the line range.
The number of rows is determined by the number of lines read from the data file.

Arguments

Inputs		
tablename	Name of table to be created	
	allowed:	string
	Default:	
asciifile	Name of ASCII file to be read	
	allowed:	string
	Default:	
headerfile	Name of an optional file defining the format	
	allowed:	string
	Default:	
autoheader	Determine header information automatically	
	allowed:	bool
	Default:	false
autoshape	Shape to be used if autoheader=True	
	allowed:	intArray
	Default:	-1
sep	Value separator	
	allowed:	string
	Default:	
commentmarker	Regex indicating comment line	
	allowed:	string
	Default:	
firstline	First line to use	
	allowed:	int
	Default:	0
lastline	Last line to use	
	allowed:	int
	Default:	-1
nomodify	Open Read-only?	
	allowed:	bool
	Default:	true
columnnames	Column Names	
	allowed:	stringArray
	Default:	
datatypes	Data types	
	allowed:	stringArray
	Default:	

Returns
bool

---

table.open.html

table.open - Function

3.2.1 open an existing table

Description

Opens a disk file containing an existing CASA Table.
Most of the time you just need to specify the tablename and perhaps nomodify.
A table can be shared by multiple processes by using the appropriate locking options. The possible options are:
- auto: let the system take care of locking. At regular time intervals these autolocks are released to give other processes the opportunity to access the table.
- autonoread: as auto, but no read locking is needed. This must be used with care, because it means that reading can be done while the table tool is not synchronized with the table file (as is normally done when a lock is acquired). The function `resync` can be used to explicitly synchronize the table tool
- user: the user takes care by explicit calls to lock and unlock
- usernoread: as user and the no readlocking behaviour of autonoread.
- permanent: use a permanent lock; the constructor fails when the table is already in use in another process
- permanentwait: as above, but wait until the other process releases its lock
- default: this is the default option. If the given table is already open, the locking option in use is not changed. Otherwise it reverts to auto.
When auto locking is used, it is possible to give a record containing the fields option, interval, and/or maxwait. In this way advanced users have full control over the locking options. In practice this is hardly ever needed.

Arguments

Inputs		
tablename		
	allowed:	string
	Default:	
lockoptions	locking dictionary to be used : dict keys are 'option', 'interval', 'maxwait'	
	allowed:	record
	Default:	
nomodify		
	allowed:	bool
	Default:	true

Returns
bool

Example

```
# First let's make a table for testing
def maketesttable():
    # Get path to CASA home directory by stipping name from '$CASAPATH'
    pathname=os.environ.get("CASAPATH").split()[0]
    # This is where the 3C273XC1.fits data should be
    fitsdata=pathname+"/data/demo/3C273XC1.fits"
    # Remove old table if present
    !rm -rf 3C273XC1.MS
    ms.fromfits("3C273XC1.MS",fitsdata)
    ms.close()

maketesttable()
tb.open("3C273XC1.MS")
tb.browse()
tb.close()
```

The first line opens an existing table 3C273XC1.MS, the second browses it using the browse function.

```
tb.open("3C273XC1.MS", nomodify=False, lockoptions={'option':'user'})
tb.lock();
tb.addrows();
tb.unlock();
```

In this example explicit user locking is used. The function lock is needed to acquire a (write) lock before the addrows is done. Thereafter the lock is released to give other processes the chance to operate on the table.
\\Note that releasing a lock implies flushing the table, so doing that very often can be quite expensive.

table.create.html

table.create - Function

3.2.1 create a new table

Description

Create a new CASA Table.
Most of the time you just need to specify the table's name and a description of its format.
A table can be shared by multiple processes by using the appropriate locking options. The possible options are:
- auto: let the system take care of locking. At regular time intervals these autolocks are released to give other processes the opportunity to access the table.
- autonoread: as auto, but no read locking is needed. This must be used with care, because it means that reading can be done while the table tool is not synchronized with the table file (as is normally done when a lock is acquired). The function `resync` can be used to explicitly synchronize the table tool
- user: the user takes care by explicit calls to lock and unlock
- usernoread: as user and the no readlocking behaviour of autonoread.
- permanent: use a permanent lock; the constructor fails when the table is already in use in another process
- permanentwait: as above, but wait until the other process releases its lock
- default: this is the default option. If the given table is already open, the locking option in use is not changed. Otherwise it reverts to auto.
When auto locking is used, it is possible to give a record containing the fields option, interval, and/or maxwait. In this way advanced users have full control over the locking options. In practice this is hardly ever needed.

Arguments

Inputs		
tablename		
	allowed:	string
	Default:	
tabledesc	description of the table's format	
	allowed:	record
	Default:	
lockoptions	locking to be used	
	allowed:	record
	Default:	default
endianformat		
	allowed:	string
	Default:	
memtype		
	allowed:	string
	Default:	
nrow		
	allowed:	int
	Default:	0
dminfo	Data Manager information	
	allowed:	record
	Default:	

Returns
bool

Example

```
# First let's get sample descriptions of a table and its data managers.
import os, shutil

def get_tabledesc_and_dminfo(tabname="3C273XC1.MS"):
    made_copy = False

    # Fetch new table if tabname not present
    if not os.path.isdir(tabname):
        # Get path to CASA root directory by stripping name from '$CASAPATH'
        pathname = os.environ.get("CASAPATH").split()[0]
```

```
            # There should be some data here
            fitsdata = pathname + "/data/demo/3C273XC1.fits"
            tabname = "3C273XC1.MS"

            ms.fromfits(tabname, fitsdata)
            ms.close()
            made_copy = True

        tb.open(tabname)
        tabdesc = tb.getdesc()
        dminfo  = tb.getdminfo()
        print tabname, "has", tb.nrows(), "rows."
        tb.close()

        # Clean up
        if made_copy:
            shutil.rmtree(tabname)

        return tabdesc, dminfo

    tabdesc, dmi = get_tabledesc_and_dminfo()
    tabdesc  # prints tabdesc
    dmi      # prints dmi

    # You could alter tabdesc and/or dmi at this point.

    # Unnecessary, but just to show there is nothing up my sleeve...
    tb.close()

    tb.create("myempty.ms", tabdesc, dminfo=dmi)
    tb.nrows()     # 0L
    tb.addrows(5)  # Add the rows _before_ filling the columns.
    tb.putcol('ARRAY_ID', numpy.array([0 for i in range(5)]))
    tb.putcol('ANTENNA1', numpy.array(range(5)))
    tb.putcol('ANTENNA2', numpy.array(range(1,6)))
    tb.browse()    # Still mostly, but not completely, empty.
    tb.close()
```

This creates a CASA table using a description of a table and its data managers from an exist

---

table.flush.html

table.flush - Function

3.2.1 flush the current contents to disk

Description

Until a flush is performed, the results of all operations are not reflected in any change to the disk file. Hence you *must* do a flush to write the changes to disk.

Arguments

Returns
bool

---

table.fromASDM.html

table.fromASDM - Function

3.2.1 Create an CASA table from an ASDM table

Description

.keywords DATE A ”07/7/23” REVISION D 0 AUTHOR A ”Paulo C. Cortes” INSTRUMENT A ”ALMA” .endkeywords
The main function for this task is to create a CASA::Table from a XML ASDM Table. The classes asdmCasaXMLUtil and asdmCasaSaxHandler are the main objects which implement the task. The asdmCasaSaxHandler encapsulate all the operations returning a reference to a CASA::Table. The class uses xerces-c to parse the XML table and creates the CASA::Table. The implementation assumes the integrity of the XML data, it not attempting to check whether the XML data meets a column format or not. In detail, an ArrayString column should agree with the following format: nd nx ... data, where nd is the number of dimensions, nx is the size of the first dimension (implemented upto a cube, i.e. nx,ny,nz), and data is the array itself which should have the appropiate number of elements. For example, a VectorString column could be: 1 2 ”I” ”Q” or dimension 1, size 2, and two string elements. Due to the lack of data type spefication in the XML tables, the column names are hardcoded into the asdmCasaSaxHandler based on the ASDM specification (see http://aramis.obspm.fr/∼alma/ASDM/ASDMEntities/index.html).
While missing data from a table column will be accepted by the task, any new column beyond the specification has to be added into the class, also, any change in data types form the specificatin will produce a crash, CASA is picky with data types integrity. So far, the list of tables included in the class is: AlmaCorrelatorMode.xml, Antenna.xml ConfigDescription.xml, DataDescription.xml, ExecBlock.xml, Feed.xml, Field.xml, Main.xml, Polarization.xml, Processor.xml, Receiver.xml, SBSummary.xml, Scan.xml, Source.xml, SpectralWindow.xml, State.xml, Station.xml, Subscan.xml, SwitchCycle.xml, CalCurve.xml, CalData.xml, CalPhase.xml
more tables will follow. The usage of fromASDM is simple, it gets two string, tablename and xmlfile, where tablename is the CASA::Table to be written and xmlfile represents the ASDM XML table. To call it do:
tb.fromasdm(tablename,xmlfile)

Arguments

Inputs		
tablename	Name of table to be created	
	allowed:	string
	Default:	
xmlfile	Name of the XML file to be read	
	allowed:	string
	Default:	

Returns
bool

---

table.resync.html

table.resync - Function

3.2.1 resync the table tool with table file

Description

Acquiring a read or write lock automatically synchronizes the internals of the table tool with the actual contents of the table files. In this way different processes accessing the same table always use the same table data.
However, a table can be used without read locking. In that case the table tool internals are not synchronized automatically. The resync function offers a way to do explicit synchronization. It is only useful if the table is opened with locking mode `autonoread` or `usernoread`.

Arguments

Returns
bool

---

table.close.html

table.close - Function

3.2.1 close the table tool

Description

First a flush is done, then the table is closed inside casapy and is no longer available for use.

Arguments

Returns
bool

---

table.copy.html

table.copy - Function

3.2.1 copy a table

Description

Copy the table. All subtables are also copied. References to another table are preserved.
The argument `deep` determines how a reference table (i.e. the result of a query) is copied. By default a file copy is made, thus the resulting table still contains references and no actual data. If, however, `deep=True` is given, a deep copy is made which means that the actual data are copied. Also all subtables are copied.
Normally a plain table is copied by copying the files. However, if `deep=True` and `valuecopy=True` are given, a plain table is copied by copying all its values and subtables. This is useful to reorganize the tables, i.e. to regain file space that is wasted by frequent updates to a table.
The argument `dminfo` can be used to specify explicit data manager info for the columns in the new plain table. It can be used to change, for example, a storage manager from IncrStMan to StandardStMan. The `dminfo` is a record as returned by the getdminfo If `dminfo` is a non-empty record, it forces `valuecopy=True`.
The standard operation is make the copy to a plain table. It is, however, possible to copy to a memory table by giving `memorytable=True`.
The endian format for the newly created table can be specified. This is only meaningful if a deep copy is made to a plain table. The possible values are:
- big: big endian format (as used on e.g. SUN)
- little: little endian format (as used on e.g. PC)
- local: use the endian format of the machine being used
- aipsrc: use the endian format specified in aipsrc variable table.endianformat (which defaults to big).
The default is aipsrc.
Normally the `copy` function only copies the table and does not create a new table tool object. The user can do that by opening the newly created table in the standard way. However, it is possible to get an object back by using `returnobject=True`. An object is always returned if the copy is made to a memory table.

Arguments

Inputs		
newtablename	Name of newtable on disk	
	allowed:	string
	Default:	
deep	Make a deep copy of a reference table?	
	allowed:	bool
	Default:	false
valuecopy	Make a deep copy of any table?	
	allowed:	bool
	Default:	false
dminfo	Data manager info for new table	
	allowed:	record
	Default:	
endian	Endian format of new table	
	allowed:	string
	Default:	aipsrc
memorytable	Hold new table in memory?	
	allowed:	bool
	Default:	false
returnobject	Return a tool object for the new table	
	allowed:	bool
	Default:	false
norows	Don't copy any rows (useful for copying only the table structure)	
	allowed:	bool
	Default:	false

Returns
table

---

table.copyrows.html

table.copyrows - Function

3.2.1 copy rows from this table to another

Description

Copy rows from this table to another. By default all rows of this table are appended to the output table. It is possible though to control which rows are copied.
Rows are added to the output table as needed. Because no rows can be added to a reference table, it is only possible to overwrite existing rows in such tables. Only the data of columns existing in both tables will be copied. Thus by making a reference table consisting of a few columns, it is possible to copy those columns only.

Arguments

Inputs		
outtable	table object of output table	
	allowed:	string
	Default:	
startrowin	First row to take from input table	
	allowed:	int
	Default:	0
startrowout	First row to write in output table, -1 (=end)	
	allowed:	int
	Default:	-1
nrow	Nr of rows to copy, -1 (=all)	
	allowed:	int
	Default:	-1

Returns
bool

Example

This example appends rows to the table itself, thus doubles the number of rows.

```
tb.open('3C273XC1.MS',nomodify=False)
tb.copyrows('3C273XC1.MS')
tb.close()
```

This example copies 10 rows of the selected subset of the MS to the beginning of the output MS.

```
!rm -rf in.MS out.MS
ms.fromfits('in.MS','3C273XC1.fits')       #Make two MSs
ms.fromfits('out.MS','3C273XC1.fits')      #for example
ms.close()
tb.open("in.MS")
t1 = tb.query('ANTENNA1==0')
tb.close()
t1.copyrows("out.MS",nrow=10,startrowout=0)
t1.close()
```

---

table.done.html

table.done - Function

3.2.1 end the table tool

Description

Effectively a synonym for function close.

Arguments

Returns
bool

---

table.iswritable.html

table.iswritable - Function

3.2.1 is the table writable?

Description

Test if the table is opened for write.

Arguments

Returns
bool

---

table.endianformat.html

table.endianformat - Function

3.2.1 get the endian format used for this table

Description

Get the endian format used for this table. It returns a string with value 'big' or 'little'.

Arguments

Returns
string

---

table.lock.html

table.lock - Function

3.2.1 acquire a lock on the table

Description

Try to acquire a read or write lock on the table. Nothing will be done if the table is already correctly locked by this process. It is only needed when user locking is used. When the lock is acquired, the internal caches will be synchronized with the (possibly changed) contents of the table.
It is possible to specify the number of attempts to do (1 per second) in case the table is locked by another process. The default 0 is trying indefinitely.

Arguments

Inputs		
write	Write lock? (F=read lock)	
	allowed:	bool
	Default:	true
nattempts	Nr of attempts	
	allowed:	int
	Default:	0

Returns
bool

---

table.unlock.html

table.unlock - Function

3.2.1 unlock and flush the table

Description

The table is flushed and the lock on the table is released. This function is only needed when user locking is used. However, it is also possible to use it with auto locking. In that case the lock will automatically be re-acquired before the next table operation.

Arguments

Returns
bool

---

table.datachanged.html

table.datachanged - Function

3.2.1 has data changed in table?

Description

This function tests if data in the table have changed (by another process) since the last call to this function.

Arguments

Returns
bool

---

table.haslock.html

table.haslock - Function

3.2.1 has this process a lock on the table?

Description

Has this process a read or write lock on the table?

Arguments

Inputs		
write	Has it a write lock? (F=read lock)	
	allowed:	bool
	Default:	true

Returns

bool

---

table.lockoptions.html

table.lockoptions - Function

3.2.1 get the lock options used for this table

Description

Get the lock options used for this table. It returns a record with the fields: option, interval and maxwait. The record can be used as the lockoptions argument when opening a table.

Arguments

Returns
record

---

table.ismultiused.html

table.ismultiused - Function

3.2.1 is the table in use in another process?

Description

Is the table still in use in another process? If so, the table cannot be deleted.

Arguments

Inputs		
checksubtables	check if subtables are multiused?)	
	allowed:	bool
	Default:	false

Returns

bool

---

table.browse.html

table.browse - Function

3.2.1 browse a table using a graphical browser

Description

To start the browser, the environment variable DISPLAY must be set.

Arguments

Returns
bool

---

table.name.html

table.name - Function

3.2.1 return name of table on disk

Description

Gives the name of the CASA table on disk that the table tool has open.

Arguments

Returns
string

Example

```
  tb.open("3C273XC1.MS")
  tb.name()
# 3C273XC1.MS
```

---

table.createmultitable.html

table.createmultitable - Function

3.2.1 Create a virtually concatenated table

Description

Arguments

Inputs	
outputTableName	name of the concatenated table allowed: string Default:
tables	list of the names of the tables to be concatenated allowed: stringArray Default:
subdirname	optional name of the subdirectory into which the input tables are moved allowed: string Default:

Returns
bool

Example

table.toasciifmt.html

table.toasciifmt - Function

3.2.1 Write CASA table into an ASCII format

Description

Write a table into an ASCII format approximately compatible with fromascii except that in order to permit variable shaped arrays (as they often occur in MSs), array values are output enclosed in square brackets. The separator between values can be specified and defaults to a blank. Note that columns containing invalid data or record type data are ignored and a warning is issued. If the argument headerfile is set then the header information is written to that file instead of the first two lines of the data file.

Arguments

Inputs	
asciifile	Name of ASCII file to be written
	allowed: string
	Default:
headerfile	Name of an optional file defining the format
	allowed: string
	Default:
columns	Names of columns to be written, default is all
	allowed: stringArray
	Default:
sep	Value separator, default is one blank
	allowed: string
	Default:

Returns
bool

Example

```
tb.toasciifmt(asciifile='myfile3.dat', headerfile='myfile3.head', columns=['SOURCE_ID', 'NAM
```

will produce a comma separated ASCII output of the three columns 'SOURCE_ID', 'NAME', and 'F
and a format description in 'myfile3.head'.

```
tb.toasciifmt(asciifile='myfile.dat')
```

will produce a space separated ASCII output of all table columns into file 'myfile.dat'
with the first two lines containing a format description.

---

table.taql.html

table.taql - Function

3.2.1 Make a table from a TaQL command.

Description

This method Expose TaQL to the user. Details on TaQL maybe found at
http://www.astron.nl/aips++/docs/notes/199

Arguments

Inputs		
taqlcommand	TaQL expression	
	allowed:	string
	Default:	TaQL expression

Returns

table

Example

```
For more information on TaQL see http://www.astron.nl/aips++/docs/notes/199
```

---

table.query.html

table.query - Function

3.2.1 Make a table from a query

Description

Make a table from a query applied to the current table. It is possible to specify column(s) and/or expressions to sort on and to specify the columns to be contained in the output table. See the example below. A new "on-the-fly" table tool is returned. The new (reference) table can be given a name and will then be written to disk. Note that the resulting table is just a reference to the original table. One can make a deep copy of the query result using the copy function (see example).

Arguments

Inputs		
query	Query string	
	allowed:	string
	Default:	String
name	Name of resulting reference table	
	allowed:	string
	Default:	
sortlist	Sort string (one or more expressions separated by commas)	
	allowed:	string
	Default:	
columns	List of column names separated by commas	
	allowed:	string
	Default:	
style	How to handle numeric ranges and order axes	
	allowed:	string
	Default:	

Returns
table

Example

```
  tb.open("3C273XC1.MS")
  subt=tb.query("OBSERVATION_ID==0",
                sortlist="ARRAY_ID", columns="TIME, DATA, UVW")
  print subt.ncols()
# 23
  tb.close()
  copyt = subt.copy ("3C273XC1_spw1.MS", True)
  subt.close()
  copyt.close()
```

From the original table corresponding to the disk file 3C273XC1.MS, only rows with OBSERVATION_ID equal to 0 are selected and sorted by ARRAY_ID. Only the columns TIME DATA UVW are written. Thereafter a deep copy of the result is made. This table query command is equivalent to the Table Query Language (TaQL) command

```
  SELECT TIME, DATA, UVW
  FROM 3C273XC1.MS
  WHERE OBSERVATION_ID==0
  ORDERBY ARRAY_ID
```

See http://www.astron.nl/casacore/trunk/casacore/doc/notes/199.html for an explanation of Ta

If "style" is not blank, "using style <style> " is prepended to the query. See http://www.astron.nl/casacore/trunk/casacore/doc/notes/199.html#x1-50002.2 for an explanation and list of choices for style. The default (glish) style is 1-based, inclusive end, and Fortran ordering. You may prefer python (0-based, exclusive end, and C order) style.

```
tb.open('any_data')
tsel = tb.selectrows([0])
print tsel.nrows() # returns 1
tsel = tb.query('ROWNUMBER()==0')
print tsel.nrows() # returns 0
tsel = tb.query('ROWNUMBER()==0', style='python')
print tsel.nrows() # returns 1
tb.close()
```

Note that style had no effect on the "OBSERVATION_ID==0" query above.

Example

The sortlist argument can be used to sort in ascending or descending order (or a mix of them) on one or more columns. Default is ascending. It is also possible to remove duplicate values using the word NODUPLICATES at the beginning.

E.g.:

```
sortlist='TIME desc'
sortlist='noduplicates ANTENNA1,ANTENNA2'
sortlist='ANTENNA1 desc, ANTENNA2 asc'
sortlist='desc ANTENNA1, ANTENNA2, TIME'
```

---

table.calc.html

table.calc - Function

3.2.1 TaQL expression with calc to calculate an expression on a table

Description

Get the result from the calculation of an expression on a table
The expression can be any expression that can be given in the WHERE clause of a SELECT expression (thus including subqueries). The given expression determines if the result is a scalar, a vector, or a record containing arrays. See the examples below.

Arguments

Inputs		
expr	Expression string	
	allowed:	string
	Default:	
prefix	TaQL prefix for style and ordering etc ...check TaQL note 199 for usage	
	allowed:	string
	Default:	using style base0, endincl, fortranorder
showtaql	Show the full taql command used	
	allowed:	bool
	Default:	false

Returns
anyvariant

Example

```
tb.calc('[select from ngc5921.ms  giving [mean(abs(DATA))]]')

find the mean of the abs of each row of the  DATA column of the MeasurementSet ngc5921.ms
```

```
returns a (potentially enormous) record where a field contains the
value of the expression for the row with that number. Note that it
returns a record because for each row the expression results in an array.
It should be clear that this example is useless. However, something
like this could be useful for a column with (very) small arrays.

tb.calc('[select from ngc5921.ms.contsub giving  [ntrue(FLAG)]]')


returns for each row the number of flags set. The result is a vector,
because for each row the expression results in a scalar.


tb.calc('sum([select from ngc5921.ms.contsub giving  [ntrue(FLAG)]])')

returns the total number of flags set in the table (in a single
scalar).

using subrow array
tb.calc('median([select from ngc5921.ms where ANTENNA1==3 && ANTENNA2==5 giving [abs(DATA[0,

The above will find the median channel 31 and 0th pol of the requested  baseline formed with
Note that the that the default casa order of arrays is fortran order ...pol axis is before c

tb.calc('median([select from ngc5921.ms where ANTENNA1==3 && ANTENNA2==5 giving [abs(DATA[31

Now the same is as the above but using the python style of axis ordering access
```

table.selectrows.html

table.selectrows - Function

3.2.1 Make a table from a selection of rows

Description

Create a (reference) table containing a given subset of rows. It is, for instance, useful when a selection is done on another table containing the row numbers in the main table. It can be useful to apply the casapy function unique to those row numbers, otherwise the same row might be included multiple times (see example).
It is possible to give a name to the resulting table. If given, the resulting table is made persistent with that table name. Otherwise the table is transient and disappears when closed or when casapy exits.
The rownumbers function returns a vector containing the row number in the main table for each row in the selection table. Thus given a row number vector `rownrs`, the following is always true.

```
rownrs == tb.selectrows(rownrs).rownumbers()
```

However, it is not true when selectrows is used on a selection table. because `rownumbers` does not return the row number in that selection table but in the main table.
It means that one has to take great care when using `selectrows` on a selection table.

Arguments

Inputs		
rownrs	0-based Row Numbers	
	allowed:	intArray
	Default:	
name	Name of resulting table	
	allowed:	string
	Default:	

Returns

table

Example

```
# EXAMPLE NOT VERIFIED SINCE query IS BROKEN
#   Do the query on the main table.
  tb.open('SOMENAME')
  scantable = tb.query(command)
#   Get the column containing the 0-based row numbers in the BACKEND table.
#   Make the row numbers unique.  NEED TO REPLACE GLISH unique FUNCTION HERE!
  backrows = unique(scantable.getcol('NS_GBT_BACKEND_ID'))
#   Form the table subset of the BACKEND table containing those rows.
  tb.close()
  tb.open('SOMENAME/GBT_BACKEND')
  scanback = tb.selectrows(backrows);
#   Do something with that table.
  print scanback.nrows();
```

---

table.info.html

table.info - Function

3.2.1 get the info record

Description

The info record contains information on the table.

Arguments

Returns
record

---

table.putinfo.html

table.putinfo - Function

3.2.1 set the info record

Description

The info record contains information on the table. It is written by applications, and used to determine what type of information is stored in a table.

Arguments

Inputs		
value	Info record	
	allowed:	record
	Default:	

Returns

bool

---

table.addreadmeline.html

table.addreadmeline - Function

3.2.1 add a readme line to the info record

Description

A readme line is part of the info record associated with a table. It is to inform the user, and is not used by any application directly.

Arguments

Inputs		
value	readme line	
	allowed:	string
	Default:	

Returns

bool

---

table.summary.html

table.summary - Function

3.2.1 summarize the contents of the table

Description

A (terse) summary of the table contents is sent to the defaultlogger.

Arguments

Inputs		
recurse	Summarize subtables recursively	
	allowed:	bool
	Default:	false

Returns

bool

Example

```
  tb.open("tcal")
  tb.summary()
# successful nomodify open of table  tcal :  9 columns, 11 rows
# Table summary: tcal
# Shape: 9 columns by 11 rows
# Info: [type=Calibration, subType=T Jones, readme=]
# Table keywords: [Type=T Jones, Interval=30, DeltaT=1]
# Columns: StartTime StopTime Gain SolutionOK Fit FitWeight
# iSolutionOK iFit iFitWeight
```

table.colnames.html

table.colnames - Function

3.2.1 return the names of the columns

Description

The names of the columns in the table are returned as a vector of Strings.

Arguments

Returns
stringArray

Example

```
  tb.open("tcal")
  tb.colnames()
# StartTime StopTime Gain SolutionOK Fit FitWeight iSolutionOK iFit iFitWeight
```

---

table.rownumbers.html

table.rownumbers - Function

3.2.1 !!!INPUT PARAMETERS IGNORED!!! return the row numbers in the (reference) table

Description

!!!NOTE INPUT PARAMETERS IGNORED!!!
This function can be useful after a selection or a sort. It returns the row numbers of the rows in this table with respect to the given table. If no table is given, the original table is used.
For example:

```
!!!NOTE INPUT PARAMETERS IGNORED!!!

  tb.open('3C273XC1.MS')
  t1=tb.selectrows([1,3,5,7,9])
  t1.rownumbers()
# [1L, 3L, 5L, 7L, 9L]
  t2=t1.selectrows([2,4])
  t2.rownumbers(t1)
# [2L, 4L]
  t2.rownumbers(tb.name())
# [5L, 9L]
  t2.rownumbers()
# [5L, 9L]
```

The last statements show that the function returns the row numbers referring to the given table. Table t2 contains rows 2 and 4 in table t1, which are rows 5 and 9 in table '3C273XC1.MS'.
Note that when a table is opened using its name, that table can be a reference table. Thus in the example above the last 2 statements may give different results depending on the fact if 3C273XC1.MS is a reference table or not.
The function should always be called with a table argument. The ability of omitting the argument is only present for backward compatibility.
The function can be useful to get the correct values from the result of a getcol or getcolslice on the original table.
!!!NOTE INPUT PARAMETERS IGNORED!!!

Arguments

Inputs		
tab	Table to which the row numbers refer	
	allowed:	record
	Default:	
nbytes	Maximum cache size in bytes	
	allowed:	int
	Default:	0

Returns
intArray

Example

```
!!!NOTE INPUT PARAMETERS IGNORED!!!
  tb.open("3C273XC1.MS")
  tb.nrows()
#7669L
  data=tb.getcolslice("DATA", [0,0], [0,0])
  data.shape
#(1, 1, 7669)
  selt=tb.query("ANTENNA1==1")
  selt.nrows()
#544L
  print len(selt.rownumbers())
#544L
```

table.setmaxcachesize.html

table.setmaxcachesize - Function

3.2.1 set maximum cache size for column in the table

Description

It can sometimes be useful to limit the size of the cache used by a column stored with the tiled storage manager. This function requires some more knowledge about the table system and is not meant for the casual user.

Arguments

Inputs	
columnname	Name of column
	allowed: string
	Default:
nbytes	Maximum cache size in bytes
	allowed: int
	Default:

Returns

bool

Example

```
  tb.open("3C273XC1.MS")
  tb.nrows()
# 7669L
  tb.setmaxcachesize ("DATA", 4*1024*1024);
# True
```

table.isscalarcol.html

table.isscalarcol - Function

3.2.1 is the specified column scalar?

Description

A column may contain either scalars or arrays in each cell. This tool function tests if the specified column has scalar contents.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	

Returns

bool

Example

```
  tb.open("tcal")
  tb.isscalarcol("StartTime")
# True
  tb.open("tcal")
  tb.isscalarcol("Gain")
# False
```

table.isvarcol.html

table.isvarcol - Function

3.2.1 tell if column contains variable shaped arrays

Description

This functions tells if the column contains variable shaped arrays. If so, the function `getvarcol` should be used to get the entire column. Otherwise `getcol` can be used.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	

Returns

bool

---

table.coldatatype.html

table.coldatatype - Function

3.2.1 return the column data type

Description

A column may contain various data types. This tool function returns the type of the column as a string.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	

Returns

string

Example

```
  tb.open("tcal")
  tb.coldatatype("StartTime")
# double
  tb.open("tcal")
  tb.coldatatype("Gain")
# complex
```

---

table.colarraytype.html

table.colarraytype - Function

3.2.1 return the column array type

Description

The possible column array types are defined as:

FixedShape FixedShape means that the shape of the array must be the same in each cell of the column. If not given, the array shape may vary. Option Direct forces FixedShape.

Direct Direct means that the data is directly stored in the table. Direct forces option FixedShape. If not given, the array is indirect, which implies that the data will be stored in a separate file.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	

Returns

string

Example

```
  tb.open("tcal")
  tb.colarraytype("Gain")
# Direct,FixedShape
```

table.ncols.html

table.ncols - Function

3.2.1 return number of columns

Arguments

Returns
int

Example

```
  tb.open("3C273XC1.MS")
  tb.ncols()
# 23L
```

---

table.nrows.html

table.nrows - Function

3.2.1 return number of rows

Description

Note that rows are numbered starting at 0.

Arguments

Returns
int

Example

```
  tb.open("3C273XC1.MS")
  tb.nrows()
# 7669L
```

---

table.addrows.html

table.addrows - Function

3.2.1 add a specified number of rows

Description

Rows can be added to the end of a table that was opened nomodify=False. The new rows are empty.

Arguments

Inputs		
nrow	Number of rows to add	
	allowed:	int
	Default:	1

Returns

bool

---

table.removerows.html

table.removerows - Function

3.2.1 remove the specified rows

Description

Remove the row numbers specified in the vector from the table. It fails when the table does not support row removal.

Arguments

Inputs		
rownrs	Row numbers to remove	
	allowed:	intArray
	Default:	

Returns

bool

---

table.addcols.html

table.addcols - Function

3.2.1 !!!REQUIRES COLUMN DESCRIPTION FUNCTIONS THAT HAVE NOT BEEN IMPLEMENTED!!! add one or more columns

Description

Columns can be added to a table that was opened nomodify=False. The new columns will be filled with a default value (0 or blank).
!!!THESE COLUMN DESCRIPTION FUNCTIONS HAVE NOT BEEN IMPLEMENTED!!!
For each column to be added a column description has to be setup using function tablecreatescalarcoldesc or tablecreatearraycoldesc. When multiple columns are used, they have to be combined in a single record using tablecreatedesc.
It is possible to specify data manager info in order to define a data manager (storage manager or virtual column engine) for the columns to be added.

Arguments

Inputs		
desc	Description of one or more columns	
	allowed:	record
	Default:	
dminfo	Optional description data manager to use	
	allowed:	record
	Default:	

Returns
bool

Example

```
!!!REQUIRES COLUMN DESCRIPTION FUNCTIONS THAT HAVE NOT BEEN IMPLEMENTED!!!
  tb.open("mytable", nomodify=False)
```

```
  dc3=tablecreatescalarcoldesc('C3', 'a')
  dc4=tablecreatescalarcoldesc('C4', as_float(0))
  dc5=tablecreatearraycoldesc('C5', as_double(0), 2, [10,20])
  tb.addcols(dc3)
# True
  tb.addcols(tablecreatedesc(dc4, dc5))
# True

A single column can be added as such, but multiple columns have
to be combined.
```

---

table.renamecol.html

table.renamecol - Function

3.2.1 rename a column

Description

A column can be renamed in a table that was opened nomodify=False. However, renaming is not possible in a (reference) table resulting from a select or sort operation.

Arguments

Inputs		
oldname	name of column to be renamed	
	allowed:	string
	Default:	
newname	new name of column	
	allowed:	string
	Default:	

Returns

bool

Example

```
  tb.open("3C273XC1.MS", nomodify=False)
  tb.renamecol ('DATA', 'DATA2')
# T
  print tb.colnames()
  tb.renamecol ('DATA2', 'DATA')
# T
  print tb.colnames()
```

```
Column \texttt{DATA} is renamed to \texttt{DATA2} and then back to
\texttt{DATA} again..
```

table.removecols.html

table.removecols - Function

3.2.1 remove one or more columns

Description

Columns can be removed from a table that was opened nomodify=False.
It may not always be possible to remove a column, because some data managers do not support column removal. However, if all columns of a data manager are removed, it will always succeed. It results in the removal of the entire data manager (and its possible files).
Note that function getdminfo can be used to find which columns are served by which data manager.

Arguments

Inputs		
columnames	names of columns to be removed	
	allowed:	stringArray
	Default:	

Returns

bool

Example

```
  tb.open("mytable", nomodify=False)
  tb.removecols ("col1 col2")
# T
  print tb.colnames()

Two columns are removed.
```

table.iscelldefined.html

table.iscelldefined - Function

3.2.1 test if a specific cell contains a value

Description

A column containing variable shaped arrays can have an empty cell (if no array has been put into it). This function tests if a cell is defined (thus is not empty). Note that a scalar column and a fixed shape array column cannot have empty cells.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	
rownr	Row number, starting at 0	
	allowed:	int
	Default:	0

Returns

bool

---

table.getcell.html

table.getcell - Function

3.2.1 get a specific cell

Description

A cell is the value at one row in one column. It may be a scalar or an array.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	
rownr	Row number, starting at 0	
	allowed:	int
	Default:	0

Returns

anyvariant

---

table.getcellslice.html

table.getcellslice - Function

3.2.1 get a slice from a specific cell

Description

A cell is the value at one row in one column. It must be an array. The slice must be specified as blc, trc with an optional stride.
In blc and trc -1 can be used to indicate all values for a dimension (-1 in blc is equivalent to 0, so -1 is especially useful for trc).

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	
rownr	Row number, starting at 0	
	allowed:	int
	Default:	
blc	Bottom left corner (e.g. [0,0,0] is start of 3D array)	
	allowed:	intArray
	Default:	
trc	Top right corner	
	allowed:	intArray
	Default:	
incr	Stride (defaults to 1 for all axes)	
	allowed:	intArray
	Default:	1

Returns
anyvariant

Example

```
  tb.open("3C273XC1.MS")
  data=tb.getcellslice("DATA", 0, [0,0], [1,0])
  print data.shape
# [2 1]
```

---

table.getcol.html

table.getcol - Function

3.2.1 get a specific column

Description

The entire column (or part of it) is returned. Warning: it might be big! The functions can only be used if all arrays in the column have the same shape. That is guaranteed for columns containing scalars or fixed shaped arrays. For columns containing variable shaped arrays it only succeeds if all those arrays happen to have the same shape.
Note that function `getvarcol` can be used to get a column of arbitrary shaped arrays, which also handles empty cells correctly. Function `isvarcol` tells if a column contains variable shaped arrays. shaped

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	
startrow	First row to read (default 0)	
	allowed:	int
	Default:	0
nrow	Number of rows to read (default -1 means till the end)	
	allowed:	int
	Default:	-1
rowincr	Increment in rows to read (default 1)	
	allowed:	int
	Default:	1

Returns
anyvariant

Example

```
  tb.open("3C273XC1.MS")
# True
  gain=tb.getcol("DATA")
  print gain.shape
# (4, 1, 7669)
```

---

table.getvarcol.html

table.getvarcol - Function

3.2.1 get a specific column (for variable arrays)

Description

Function `getcol` can only used if values in the column cells to get have the same shape. Function `getvarcol` addresses this limitation by returning the values as a record instead of an array. Each field in the record contains the value for a column cell. If the value is undefined (i.e. the cell does not contain a value), the unset value is put in the record. Each field name is the letter r followed by the row number. The length of the record is the number of rows to get.
Note that the function `isvarcol` tells if a column contains variable shaped arrays.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	
startrow	First row to read (default 0)	
	allowed:	int
	Default:	0
nrow	Number of rows to read (default -1 means till the end)	
	allowed:	int
	Default:	-1
rowincr	Increment in rows to read (default 1)	
	allowed:	int
	Default:	1

Returns
record

Example

```
  tb.open("3C273XC1.MS")
  gain=tb.getvarcol("DATA")
  print len(gain)
# 7669
```

---

table.getcolslice.html

table.getcolslice - Function

3.2.1 get a slice from a specific columnarray

Description

A slice from the entire column (or part of it) is returned. Warning: it might be big!
In blc and trc -1 can be used to indicate all values for a dimension (-1 in blc is equivalent to 1, so -1 is especially useful for trc). Note that blc and trc should not contain the row number, only the blc and trc of the arrays in the column.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	
blc	Bottom left corner (e.g. [0,0,0] is start of 3D array)	
	allowed:	intArray
	Default:	
trc	Top right corner	
	allowed:	intArray
	Default:	
incr	Stride (defaults to 1 for all axes)	
	allowed:	intArray
	Default:	
startrow	First row to read (default 0)	
	allowed:	int
	Default:	0
nrow	Number of rows to read (default -1 means till the end)	
	allowed:	int
	Default:	-1
rowincr	Increment in rows to read (default 1)	
	allowed:	int
	Default:	1

Returns

anyvariant

Example

```
  tb.open("3C273XC1.MS")
  data=tb.getcolslice("DATA", [0,0], [1,0])
  data.shape
# (2 1 7669)
```

---

table.putcell.html

table.putcell - Function

3.2.1 put a specific cell

Description

A cell is the the value at one row in one column. It may be a scalar or an array.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	
rownr	Row number(s) (0-relative)	
	allowed:	intArray
	Default:	
thevalue	Value	
	allowed:	any
	Default:	variant

Returns

bool

---

table.putcellslice.html

table.putcellslice - Function

3.2.1 put a slice into a specific cell

Description

A cell is the value at one row in one column. It must be an array. The slice must be specified as blc, trc with an optional stride.
In blc and trc -1 can be used to indicate all values for a dimension (-1 in blc is equivalent to 0, so -1 is especially useful for trc).

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	
rownr	Row number, starting at 0	
	allowed:	int
	Default:	
value	Value	
	allowed:	any
	Default:	variant
blc	Bottom left corner (e.g. [0,0,0] is start of 3D array)	
	allowed:	intArray
	Default:	
trc	Top right corner	
	allowed:	intArray
	Default:	
incr	Stride (defaults to 1 for all axes)	
	allowed:	intArray
	Default:	1

Returns
bool

---

table.putcol.html

table.putcol - Function

3.2.1 put a specific column

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	
value	Array	
	allowed:	any
	Default:	variant
startrow	First row to put (default 0)	
	allowed:	int
	Default:	0
nrow	Number of rows to put (default -1 means till the end)	
	allowed:	int
	Default:	-1
rowincr	Increment in rows to put (default 1)	
	allowed:	int
	Default:	1

Returns

bool

Example

```
tb.open("3C273XC1.MS",nomodify=False)
data=tb.getcol("DATA")
# [could modify data here]
tb.putcol("DATA", data)
tb.flush()
```

table.putvarcol.html

table.putvarcol - Function

3.2.1 put a specific column (for variable arrays)

Description

`putcol` can only used if values in the column cells to put have the same shape. `putvarcol` addresses this limitation by passing the values as a record instead of an array. Each field in the record contains the value for a column cell. So the length of the record has to match the number of rows to put. If a value is the unset value, no put is done for that row.

Arguments

Inputs	
columnname	Name of column
	allowed: string
	Default:
value	Record with values
	allowed: record
	Default:
startrow	First row to put (default 0)
	allowed: int
	Default: 0
nrow	Number of rows to put (default -1 means till the end)
	allowed: int
	Default: -1
rowincr	Increment in rows to put (default 1)
	allowed: int
	Default: 1

Returns

bool

Example

```
  tb.open("3C273XC1.MS",nomodify=False)
  gain=tb.getvarcol("DATA", 0, 10)
  tb.putvarcol("Gain", gain, 10, 10)
  tb.flush()
```

This example copies the values from row 0-9 to row 10-19.

---

table.putcolslice.html

table.putcolslice - Function

3.2.1 put a slice into a specific column

Description

In blc and trc, -1 can be used to indicate all values for a dimension (-1 in blc is equivalent to 0, so -1 is especially useful for trc). Note that blc and trc should not contain the row number, only the blc and trc of the arrays in the column.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	
value	Array	
	allowed:	any
	Default:	variant
blc	Bottom left corner (e.g. [0,0,0] is start of 3D array)	
	allowed:	intArray
	Default:	
trc	Top right corner	
	allowed:	intArray
	Default:	
incr	Stride (defaults to 1 for all axes)	
	allowed:	intArray
	Default:	1
startrow	First row to put (default 0)	
	allowed:	int
	Default:	0
nrow	Number of rows to put (default -1 means till the end)	
	allowed:	int
	Default:	-1
rowincr	Increment in rows to put (default 1)	
	allowed:	int
	Default:	1

Returns

bool

Example

```
tb.open("3C273XC1.MS",nomodify=False)
data_all=tb.getcolslice("DATA", [-1,-1], [-1,=1])
print data_all.shape
# (4, 1, 7669)
data=tb.getcolslice("DATA", [0,0],[3,0])
# can modify data here
tb.putcolslice("DATA", data, [0,0],[3,0])
tb.flush()
```

---

table.getcolshapestring.html

table.getcolshapestring - Function

3.2.1 get shape of arrays in a specific column

Description

The shapes of the arrays in the entire column (or part of it) are returned as strings like [20,3]. When the column contains fixed shaped arrays, a single string is returned. Otherwise a vector of strings is returned.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	
startrow	First row to read (default 0)	
	allowed:	int
	Default:	0
nrow	Number of rows to read (default -1 means till the end)	
	allowed:	int
	Default:	-1
rowincr	Increment in rows to read (default 1)	
	allowed:	int
	Default:	1

Returns
stringArray

Example

```
tb.open("3C273XC1.MS")
shapes=tb.getcolshapestring("DATA"))
print len(shapes)
```

table.getkeyword.html

table.getkeyword - Function

3.2.1 get value of specific table keyword

Description

The value of the given table keyword is returned. The value can be of any type, including a record and a table.
If a keyword is a table, its value is returned as a string containing the table name prefixed by 'Table: '.
It is possible that the value of a keyword is a record itself (arbitrarily deeply nested). A field in such a subrecord can be read by separating the name with dots.

Arguments

Inputs		
keyword	Name or seqnr of keyword: string or int	
	allowed:	any
	Default:	variant

Returns

anyvariant

Example

```
  tb.open('3C273XC1.MS')
  tb.getkeywords()
  tb.getkeyword('MS_VERSION')
# 2.0
  tb.close()
  tb.open('tcal')
  tb.getkeyword('rec.fld')     # get field from a record
# 3.14
```

table.getkeywords.html

table.getkeywords - Function

3.2.1 get values of all table keywords

Description

The values of all table keywords are returned. The values can be of any type, including a record and a table.
If a keyword is a table, its value is returned as a string containing the table name prefixed by 'Table: '.

Arguments

Returns
record

Example

```
  tb.open('3C273XC1.MS')
  tb.getkeywords()
#{'ANTENNA': 'Table: /home/aips2mgr/testing/3C273XC1.MS/ANTENNA',
# 'DATA_DESCRIPTION': 'Table: /home/aips2mgr/testing/3C273XC1.MS/DATA_DESCRIPTION',
# 'FEED': 'Table: /home/aips2mgr/testing/3C273XC1.MS/FEED',
# 'FIELD': 'Table: /home/aips2mgr/testing/3C273XC1.MS/FIELD',
# 'FLAG_CMD': 'Table: /home/aips2mgr/testing/3C273XC1.MS/FLAG_CMD',
# 'HISTORY': 'Table: /home/aips2mgr/testing/3C273XC1.MS/HISTORY',
# 'MS_VERSION': 2.0,
# 'OBSERVATION': 'Table: /home/aips2mgr/testing/3C273XC1.MS/OBSERVATION',
# 'POINTING': 'Table: /home/aips2mgr/testing/3C273XC1.MS/POINTING',
# 'POLARIZATION': 'Table: /home/aips2mgr/testing/3C273XC1.MS/POLARIZATION',
# 'PROCESSOR': 'Table: /home/aips2mgr/testing/3C273XC1.MS/PROCESSOR',
# 'SOURCE': 'Table: /home/aips2mgr/testing/3C273XC1.MS/SOURCE',
```

```
# 'SPECTRAL_WINDOW': 'Table: /home/aips2mgr/testing/3C273XC1.MS/SPECTRAL_WINDOW',
# 'STATE': 'Table: /home/aips2mgr/testing/3C273XC1.MS/STATE'}
```

---

table.getcolkeyword.html

table.getcolkeyword - Function

3.2.1 get value of specific column keyword

Description

The value of the given column keyword is returned. The value can be of any type, including a record and a table.
If a keyword is a table, its value is returned as a string containing the table name prefixed by 'Table: '.
It is possible that the value of a keyword is a record itself (arbitrarily deeply nested). A field in such a subrecord can be read by separating the name with dots.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	
keyword	Name or seqnr of keyword: string or int	
	allowed:	any
	Default:	variant

Returns
anyvariant

Example

```
  tb.open("3C273XC1.MS")
  tb.getcolkeyword("UVW", "QuantumUnits")
#array(['m', 'm', 'm'],
#      dtype='|S2')
```

table.getcolkeywords.html

table.getcolkeywords - Function

3.2.1 get values of all keywords for a column

Description

The values of all keywords for the given column are returned. The values can be of any type, including a record and a table.
If a keyword is a table, its value is returned as a string containing the table name prefixed by 'Table: '.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	

Returns

anyvariant

Example

```
  tb.open("3C273XC1.MS")
  tb.getcolkeywords("UVW")
#{'MEASINFO': {'Ref': 'ITRF', 'type': 'uvw'},
# 'QuantumUnits': array(['m', 'm', 'm'],
#      dtype='|S2')}
```

---

table.putkeyword.html

table.putkeyword - Function

3.2.1 put a specific table keyword

Description

Put a table keyword. The value of the keyword can be a scalar or an array of any type or it can be a record.
It is possible to define a keyword holding a subtable. In that case a special string containing the name of the subtable will be passed to the table client.
It is possible that the value of a keyword is a record itself (arbitrarily deeply nested). A field in such a subrecord can be written by separating the name with dots. If a subrecord does not exist, an error is returned unless `makesubrecord=True` is given. In such a case intermediate records are created when needed.

Arguments

Inputs		
keyword	Name or seqnr of keyword: string or int	
	allowed:	any
	Default:	variant
value	Value of keyword	
	allowed:	any
	Default:	variant
makesubrecord	Create intermediate records	
	allowed:	bool
	Default:	false

Returns
bool

Example

```
tb.open("3C273XC1.MS", nomodify=False)
```

```
  tb.putkeyword("VERSION", "1.66")
# True
#      define ANTENNA subtable
  tb.putkeyword("ANTENNA", 'Table: 3C273XC1.MS/ANTENNA')
  tb.flush()
# True
#      write a field in a record and create subrecords when needed
  tb.putkeyword("REC.SUB.FLD", "val", True)
# True
#      write a keyword with a record value
  tb.putkeyword("REC", {'SUB': {'FLD': 'val'}})
# True

Note that the last example does the same as the previous one (assuming
that \texttt{REC} does not exist yet with other fields).
```

---

table.putkeywords.html

table.putkeywords - Function

3.2.1 !!!BROKEN!!! put multiple table keywords

Description

Put multiple table keywords. All fields in the given record are put as table keywords. The value of each field can be a scalar or an array of any type or it can be a record.
It is also possible to define a keyword holding a subtable. This can be done by giving the keyword a string value consisting of the subtable name prefixed by 'Table: '.

Arguments

Inputs		
value	Record of keyword=value pairs	
	allowed:	record
	Default:	

Returns
bool

Example

```
  tb.open('3C273XC1.MS', nomodify=False)
  kw=tb.getkeywords()
  print kw['MS_VERSION']
# 2.0
  kw['MS_VERSION']=2.1
  tb.putkeywords(kw)
# !!!BROKEN. Keywords containing float are not handled properly!!!
  tb.flush()
# True
```

table.putcolkeyword.html

table.putcolkeyword - Function

3.2.1 put a specific keyword for a column

Description

Put a keyword in the given column. The value of the keyword can be a scalar or an array of any type or it can be a record.
It is possible to define a keyword holding a subtable. In that case a special string containing the name of the subtable will be passed to the table client.
It is possible that the value of a keyword is a record itself (arbitrarily deeply nested). A field in such a subrecord can be written by separating the name with dots. If a subrecord does not exist, an error is returned unless `makesubrecord=True` is given. In such a case intermediate records are created when needed.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	
keyword	Name or seqnr of keyword,string or int	
	allowed:	any
	Default:	variant
value	Value of keyword	
	allowed:	any
	Default:	variant

Returns
bool

Example

```
tb.open("3C273XC1.MS", nomodify=False)
```

```
  ckw=tb.getcolkeyword("UVW","QuantumUnits")
  print ckw
  # modify ckw as desired
  tb.putcolkeyword("UVW","QuantumUnits",ckw)
# True
  tb.flush()
# True
```

---

table.putcolkeywords.html

table.putcolkeywords - Function

3.2.1 put multiple keywords for a column

Description

Put multiple keywords in the given column. All fields in the given record are put as column keywords. The value of each field can be a scalar or an array of any type or it can be a record.
It is also possible to define a keyword holding a subtable. This can be done by giving the keyword a string value consisting of the subtable name prefixed by 'Table: '.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	
value	Record of keyword=value pairs	
	allowed:	record
	Default:	

Returns

bool

Example

```
  tb.open("3C273XC1.MS", nomodify=False)
  kws = tb.getcolkeywords("UVW")
  kws
#{'MEASINFO': {'Ref': 'ITRF', 'type': 'uvw'},
# 'QuantumUnits': array(['m', 'm', 'm'],
#       dtype='|S2')}
  kws['MEASINFO']['Ref']='B1950'
```

```
  tb.putcolkeywords(kws)
# True
```

---

table.removekeyword.html

table.removekeyword - Function

3.2.1 remove a specific table keyword

Arguments

Inputs		
keyword	Name or seqnr of keyword: string or int	
	allowed:	any
	Default:	variant

Returns

bool

Example

```
  tb.open("3C273XC1.MS", nomodify=False)
  tb.removekeyword("MS_VERSION")
# True
  tb.flush()
# True
```

---

table.removecolkeyword.html

table.removecolkeyword - Function

3.2.1 remove a specific keyword for a column

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	
keyword	Name or seqnr of keyword: string or int	
	allowed:	any
	Default:	variant

Returns

bool

Example

```
  tb.open("3C273XC1.MS", nomodify=False)
  tb.removecolkeyword("UVW", "QuantumUnits")
# True
  tb.flush()
# True
```

table.getdminfo.html

table.getdminfo - Function

3.2.1 get the info about data managers

Description

This function returns the types and names of the data managers used. For each data manager it also returns the names of the columns served by it. The information is returned as a record containing a subrecord for each data manager. Each subrecord contains the fields TYPE, NAME and COLUMNS.

Arguments

Returns
record

Example

```
  tb.open('3C273XC1.MS')
  rec = tb.getdminfo()

Print the output record shows that the table uses 9 storage managers.
```

---

table.keywordnames.html

table.keywordnames - Function

3.2.1 get the names of all table keywords

Description

This function returns a vector of strings containing the names of all table keywords.

Arguments

Returns
stringArray

---

table.fieldnames.html

table.fieldnames - Function

3.2.1 get the names of fields in a table keyword

Description

This function returns a vector of strings containing the names of all fields in the given table keyword. It is only valid if the keyword value is a record. If no keyword name is given, the names of all table keywords are returned.

Arguments

Inputs		
keyword	keyword name	
	allowed:	string
	Default:	

Returns

stringArray

---

table.colkeywordnames.html

table.colkeywordnames - Function

3.2.1 get the names of all keywords in a column

Description

This function returns a vector of strings containing the names of all keywords in the column with the given name..

Arguments

Inputs		
columnname	column name	
	allowed:	string
	Default:	

Returns

stringArray

Example

```
tb.open('3C273XC1.MS')
tb.colkeywordnames("UVW")
```

---

table.colfieldnames.html

table.colfieldnames - Function

3.2.1 get the names of fields in a keyword in a column

Description

This function returns a vector of strings containing the names of all fields in the given keyword in the given column. It is only valid if the keyword value is a record.
If no keyword name is given, the names of all keywords in the column are returned.

Arguments

Inputs		
columnname	column name	
	allowed:	string
	Default:	
keyword	keyword name	
	allowed:	string
	Default:	

Returns

stringArray

---

table.getdesc.html

table.getdesc - Function

3.2.1 get the table description

Description

The table description is a casapy record that contains a complete description of the layout of the table (except for the number of rows).
By default the actual table description is returned (thus telling the actual shapes and data managers used). It is also possible to get the table description used when creating the table.

Arguments

Inputs		
actual	actual table description?	
	allowed:	bool
	Default:	true

Returns

record

Example

```
tb.open("3C273XC1.MS")
tb.getdesc()
```

table.getcoldesc.html

table.getcoldesc - Function

3.2.1 get the description of a specific column

Description

The column description is a casapy record that contains a complete description of the layout of a specified column (except for the number of rows). It can be used to construct a table description.

Arguments

Inputs		
columnname	Name of column	
	allowed:	string
	Default:	

Returns

record

Example

```
  tb.open("3C273XC1.MS")
  tb.getcoldesc("DATA")
#{'comment': 'The data column',
# 'dataManagerGroup': 'TiledData',
# 'dataManagerType': 'TiledShapeStMan',
# 'maxlen': 0,
# 'ndim': 2,
# 'option': 0,
# 'valueType': 'complex'}
```

table.ok.html

table.ok - Function

3.2.1 Is the table tool ok?

Description

Perform a number of sanity checks and return T if ok. Failure (returning F) is a sign of a bug.

Arguments

Returns
bool

---

table.clearlocks.html

table.clearlocks - Function

3.2.1 Clears any table lock associated with the current process

Description

Occasionally a table will be inretrievably locked to another process no matter how much closing is done. So clearLocks will unlock all the files in the table cache that use AutoLocking.

Arguments

Returns
bool

---

table.listlocks.html

table.listlocks - Function

3.2.1 Lists any table lock associated with the current process

Description

Occasionally a table will be inretrievably locked to another process no matter how much closing is done. So listLocks will list the offending tables (and unoffending ones, too), so we can figure out where the problem might be.

Arguments

Returns
bool

---

table.statistics.html

table.statistics - Function

3.2.1 Get statistics on the selected table column

Description

This function computes descriptive statistics on the table column. It returns the statistical values as a dictionary. The given column name must be a numerical column. If it is a complex valued column, the parameter complex_value defines which derived real value is used for the statistics computation.

Arguments

Inputs		
column	Column name	
	allowed:	string
	Default:	
complex_value	Which derived value to use for complex columns (amp, amplitude, phase, imag, real, imaginary)	
	allowed:	string
	Default:	
useflags	Use the data flags	
	allowed:	bool
	Default:	true

Returns
record

Example

```
tb.open("ggtau.1mm.amp.gcal")
s = tb.statistics(column="GAIN", complex_value="phase")
```

table.showcache.html

table.showcache - Function

3.2.1 show the contents of the table cache

Description

Show the contents of the table cache.

Arguments

Inputs		
verbose		
	allowed:	bool
	Default:	true

Returns

stringArray

Example

```
tb.showcache()
```

---

table.testincrstman.html

table.testincrstman - Function

3.2.1 Checks consistency of an Incremental Store Manager bucket layout

Description

Checks consistency of an Incremental Store Manager bucket layout
In case of corruption it returns False and a SEVERE msg is posted containing information about the location of the corrupted bucket

Arguments

Inputs		
column	Column name	
	allowed:	string
	Default:	

Returns

bool

Example

```
mytb = tbtool()
mytb.open('uid___A002_X841035_X203.ms.split')
mytb.testincrstman('FLAG_ROW')
```

tableplot-Tool.html

3.2.2 tableplot - Tool

Plot data from tables via plotter
Requires:

Synopsis

Description

`tableplot` is a plotting tool for general CASA tables. Table columns can be plotted against each other, and can be combined using TaQL to create expressions for derived quantities, the result of which can then be plotted. Data from more than one table can be accessed and plotted at the same time. Expressions producing arrays result in overlay plots. The default plot style is single-panel, but if an iteration axis is specified, multi-panel plots are supported. Zooming and region-based flagging is possible on single and multi panel plots. A GUI, adapted to a particular kind of table (measurement set) can call the tableplot functions and customize the generated plots.

Methods

open	Specify list of tables to operate on.
setgui	Set the GUI on or off. Can be done only once !!
savefig	Save the currently plotted image.
selectdata	Perform a TaQL based subtable selection for subsequent plots.
plotdata	Plot the result of a general TaQL expression.
iterplotstart	Initialize plotting with an iteration axis
replot	Replot all existing panels and layers.
iterplotnext	Start/Continue plotting
iterplotstop	Stop plot iterations.
markregions	Mark a rectangular region to flag
flagdata	Flag Data for selected flag regions
unflagdata	Unset flags for all regions marked using `tp.markregions()`.
locatedata	Print info about data selected using tp.markregions().
clearflags	Clear all flags in the table.
saveflagversion	Save current flags with a version name.
restoreflagversion	Restore flags from a saved flag_version.
deleteflagversion	Delete a saved flag_version.
getflagversionlist	Print out a list of saved flag_versions.
clearplot	Clear a plot.
done	Clean up the tableplot tool

tableplot.open.html

tableplot.open - Function

3.2.2 Specify list of tables to operate on.

Description

Specify a list of table names to open for plotting. All plots will operate on these tables until open is called again to change the list of tables. Returns true if tables are valid, false otherwise.

Arguments

Inputs		
tabnames	List of strings identifying Table names	
	allowed:	stringArray
	Default:	

Returns

bool

Example

```
# set the list of tables to plot from.
tp.open(tabnames=['3c273.ms','3c48.ms'])
```

---

tableplot.setgui.html

tableplot.setgui - Function

3.2.2 Set the GUI on or off. Can be done only once !!

Description

Set the GUI on or off. Can be done only once !!

Arguments

Inputs		
gui	gui=False to turn off gui	
	allowed:	bool
	Default:	false

Returns

bool

---

tableplot.savefig.html

tableplot.savefig - Function

3.2.2 Save the currently plotted image.

Description

Store the contents of the plot window in a file. The file format (type) is based on the file name, ie. the file extension given determines the format the file is saved as. The accepted formats are eps, ps, png, pdf, and svg.
Internally, this function uses the matplotlib pl.savefig function.
Note that if a full path is not given that the files will be saved in the current working directory.

Arguments

Inputs	
filename	Name the plot image is to be saved to. allowed: string Default:
dpi	Number of dots per inch (resolution) to save the image at. allowed: int Default: -1
orientation	Either landscape or portrait. Supported by the postscript format only. allowed: string Default:
papertype	Valid values are: letter, legal, exective, ledger, a0-a10 and b0-b10. This option is supported byt the postscript format only. allowed: string Default:
facecolor	Color of space between the plot and the edge of the square. Valid values are the same as those accepted by the plotcolour option of tp.plotdata(). allowed: string Default:
edgecolor	Color of the outer edge. Valid values are the same as those accepted by the plotcolour option of tp.plotdata(). allowed: string Default:

Returns
bool

---

tableplot.selectdata.html

tableplot.selectdata - Function

3.2.2 Perform a TaQL based subtable selection for subsequent plots.

Description

Specify a TaQL select string. A subtable will be generated, and passed to the plotter.

Arguments

Inputs		
taqlstring	TaQL string for selection	
	allowed:	string
	Default:	

Returns
bool

Example

```
# set the list of tables to plot from.
  tp.open(tabnames=['3c273.ms','3c48.ms'])
  tp.selectdata(taqlstring='ANTENNA1==5')
```

tableplot.plotdata.html

tableplot.plotdata - Function

3.2.2 Plot the result of a general TaQL expression.

Description

This function evaluates the specified TaQL expressions for the X and Y axes of a two-dimensional plot,extracts the resulting columns, and plots them.

1. TaQL expressions resulting in scalars are plotted as a single X-Y plot. Expressions involving Array-Columns can result in arrays which are then plotted as overlays.
2. The default mode of operation is to plot one expression against another. In the case of Array-Columns (ex. DATA column of a measurement set with each row containing an array of shape NColumns x NPolarzations), a Cross-plot mode allows plots with the x-axis representing one axes of the Array-Column. (ex. X-axis is channel number or polarization number).
3. If multiple tables are specified in tp.open(), then the TaQL expressions are applied to all tables and overlay plots are generated.
4. Plots can be made to separate panels whose locations on the plot window are user specified as a 3-tuple [nrows, ncolumns, panelnumber].
5. Multiple plots can be stacked upon each other on a panel (overplot mode).
6. Plotter options can be specifed to control appearance, plot-style, labels, etc.

Valid TaQL strings must satisfy the following conditions.

1. Each TaQL string must result in a Double scalar or array.
   `'AMPLITUDE(DATA[1,1])'` results in a Double scalar (valid).
   `'AMPLITUDE(DATA[1:2,1])'` results in a Double array (valid).
   `'MEAN(AMPLITUDE(DATA[1:2,1]))'` results in a Double scalar (valid).
   `'DATA[1,1]'` results in a Complex scalar (NOT valid).
   `'AMPLITUDE(DATA[1,1])<10'` results in a Bool scalar (NOT valid).
2. All TaQL functions resulting in Double Scalars/Arrays are allowed, except for those involving an explicit collapse axis (means,sums,etc..). Note that these functions are different from mean,sum,etc.. which are supported.

3. TaQL strings must be provided as pairs of strings, with the X-TaQL first, followed by the Y-TaQL. There are 3 cases.
   X-TaQL − Scalar, Y-TaQL − Scalar (one-to-one single plot)
   X-TaQL − Scalar, Y-TaQL − Array (one-to-many overlay plot)
   X-TaQL − Array, Y-TaQL − Array (if the shapes are the same, then a one-to-one mapping is done, otherwise only the first X-TaQL result is used for a one-to-many mapping with the Y-TaQL Array.)

4. For cross plots (for example amplitude vs channel plots in an MS), the X-TaQL must be a string containing 'CROSS'. The Y-TaQL is used to read out the data from the table, and the x-values are the column indices (channel numbers) chosen by the Y-TaQL.

Plotting options ('poption' entries) are listed below.
Default values are indicated within [ ] when present. nrows [1] : Number of rows of panels ncols [1] : Number of columns of panels panel [1] : Panel index. Must be in [1,nrows x ncols] plotcolour [1] : Plot colour. Codes for matplotlib are [0:black, 1:red, 2:green, 3:blue, 4:cyan, 5:yellow] [magenta is reserved for plotting flagged values if showflags = 1 , and cannot be set by the user as a plot colour] If the plotcolour field is left out from 'poption', colours are chosen automatically. multicolour [False] : True -¿ Each channel,pol appears in a different colour. False -¿ Data from all pols and channels appear in the same colour. Different colours appear for different layers (overplot). or data from different tables. timeplot [False] : True -¿ Turn on date/time formatting for the x-axis. overplot [False] : True -¿ Overlay on an existing plot. All layers will remain active for data editing via flagging. Labels will be those of the top-most layer. False -¿ Replace an existing plot with a new one. In the case of an existing stack of plots, the top-most layer is replaced. For example, this can be used to modify the colour of the top-most layer without creating an additional layer. py_plotsymbol [,] : Plot markers. Options for matplotlib are [[,]:pixel, [.]:point, [o]:circle, [x]:cross, [+]:plus, [$\hat{}$]:triangle up, [v]:triangle down, [¡]:triangle left, [¿]:triangle right, [-]:solid line, [–]:dashed line, [-.]:dash-dot line, [:]:dotted line, [s]:square, [D]:diamond, [d]:thin diamond, [1]:tripod down, [2]:tripod up, [3]:tripod left, [4]:tripod right, [h]:hexagon, [H]:rotated hexagon, [p]:pentagon, [—]:vertical line symbol, [_]:horizontal line symbol. markersize [10.0] : The size (in pixels) of the markers being plotted. Markers are specified by the py_plotsymbol option. For example, '+', 'o', and 'd' linewidth [2.0] The width of the lines that are drawn, lines are if the py_plotsymbol chosen is a line. For example, '-', '–', and ':'. plotrange [ ] : Only data within this specific range of values [xmin,xmax,ymin,ymax] will be plotted. Default is the data range. showflags [0] : True -¿ Plot only unflagged data. False -¿ Plot only flagged data. crossdirection [False] : Applies only with CROSS-plots on table ArrayColumns. False -¿ use column number as the x-axis (ex. channel no.). True -¿ use row number as the x-axis (ex. polarization no.). pointlabels [ ] : Data points can be annotated by supplying a list of labels. If N labels are supplied, the first N data points plotted will be

annotated. (Note that if data is edited via flagging,the points are relabeled to label the first N points.) windowsize [8.0] : horizontal size of plot window (inches) aspectratio [0.8] : aspect-ratio of the plot window (dx/dy) fontsize [12.0] : Font size of title text. Font size of x,y labels are 80% of this.
Returns true if plotting is successful, false otherwise.

Arguments

Inputs		
poption	Record of plot options	
	allowed:	record
	Default:	1 1 6 0.8 1.0
labels	List of strings : Title,Xlabel,Ylabel	
	allowed:	stringArray
	Default:	
datastr	List of TaQL strings : X,Y	
	allowed:	stringArray
	Default:	

Returns
bool

Example

```
Plot data amplitude vs uv-distance for two Measurement set tables as a
single panel plot.  Operate on channel 1 for Stokes 1 and 2 using the
DATA column.


 tp.open(tabnames=['3c273.ms','3c48.ms'])
 pop = { 'nrows':1, 'ncols':1,'panel':1}
 labels = ['Amplitude vs UVdist','uvdist','amplitude']
 xystr = ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1,1:2])']
 tp.plotdata(poption=pop,labels=labels,datastr=xystr)


TaQL strings for the above example can also be written as follows.

  xystr = ['SQRT(UVW[1]*UVW[1]+UVW[2]*UVW[2])','AMPLITUDE(DATA[1,1:2])']
```

Example

Multi-panel plotting

```
# uvdist for pol 1 and chan 1,2 on panel 211
pop = { 'nrows':2, 'ncols':1,'panel':1}
labels = ['Amplitude vs UVdist','uvdist','amplitude']
xystr = ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1,1:2])']
tp.plotdata(poption=pop,labels=labels,datastr=xystr)

# uvdist for pol 2 and chan 1,2 (overplot=1) on panel 211
pop = { 'nrows':2, 'ncols':1,'panel':1, 'overplot':True}
labels = ['Amplitude vs UVdist','uvdist','amplitude']
xystr = ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[2,1:2])']
tp.plotdata(poption=pop,labels=labels,datastr=xystr)

# uv coverage on panel 223
pop = { 'nrows':2, 'ncols':2,'panel':3, 'plotcolour':4}
labels = ['UV Coverage','u','v']
xystr = ['UVW[1]','UVW[2]','-UVW[1]','-UVW[2]']
tp.plotdata(poption=pop,labels=labels,datastr=xystr)

# amp(data[1:2,1:10]) vs channel number on panel 224
pop = { 'nrows':2, 'ncols':2,'panel':4, 'plotcolour':1}
labels = ['Amplitude vs Baseline number','baseline number','amplitude']
xystr = ['28*ANTENNA1+ANTENNA2-(ANTENNA1-1)*(ANTENNA1+2)/2',
          'AMPLITUDE(DATA[1:2,1:10])']
tp.plotdata(poption=pop,labels=labels,datastr=xystr)
```

Example

Plotting with time formatting

```
# vistime for 10 chans (timeplot=1)
pop = { 'nrows':1, 'ncols':1,'panel':1,'timeplot':True}
labels = ['Timeplot','time','amplitude']
xystr = ['TIME','AMPLITUDE(DATA[1:2,1:10])']
tp.plotdata(poption=pop,labels=labels,datastr=xystr)
```

Example

```
Cross-plots - take in a single TaQL expression involving an ArrayColumn,
and use the column numbers of each Array per row of the table as the x-axis.
In a measurement set, the DATA ArrayColumn contains 2D Arrays, each with
NCHAN columns and NPOL rows. Plotting with 'CROSS' as the X-TaQL, uses
channel numbers as the x-axis. The option 'crossdirection=True' can be used
to plot with polarization on the x-axis.
```

```
pop = { 'nrows':2, 'ncols':1,'panel':1, 'plotcolour':2}
labels = ['Amplitude vs Channel number','chan','amplitude']
xystr = ['CROSS','AMPLITUDE(DATA[1:2,1:10])']
tp.plotdata(poption=pop,labels=labels,datastr=xystr)
pop = { 'nrows':2, 'ncols':1,'panel':1, 'plotcolour':2, 'crossdirection':True}
labels = ['Amplitude vs Polarization number','pol','amplitude']
xystr = ['CROSS','AMPLITUDE(DATA[1:2,1:10])']
tp.plotdata(poption=pop,labels=labels,datastr=xystr)
```

Example

```
Individual points can be annotated by specifying the 'pointlabels' parameter.
If N labels are specified, the first N data points to be plotted, are annotated.
```

```
pop = {'nrows':1,'ncols':1, 'panel':1, 'plotcolour':1,'py_plotsymbol':'o',
       'pointlabels':[' A1',' A2',' A3',' A4',' A5',' A6',' A7',' A8',
                      ' A9',' B1',' B2',' B3',' B4',' B5',' B6',' B7',
                      ' B8',' B9',' C1',' C2',' C3',' C4',' C5',' C6',
```

```
                      ' C7',' C8',' C9',' D1',' D2',' D3',' D4']}
xystr = ['POSITION[1]','POSITION[2]']
labels = ['Antenna positions','x','y']
tp.plotdata(poption=pop,labels=labels,datastr=xystr)
```

Example

To plot with multiple colours for each channel and polarization of an MS.

```
#(multicolour=1, plotcolour > 5)
pop = { 'nrows':1, 'ncols':1,'panel':1, 'plotcolour':6,
        'showflags':0, 'multicolour':True}
labels = ['Amplitude vs UVdist','uvdist','amplitude']
xystr = ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1:2,1:2])']
tp.plotdata(poption=pop,labels=labels,datastr=xystr)
```

---

tableplot.iterplotstart.html

tableplot.iterplotstart - Function

3.2.2 Initialize plotting with an iteration axis

Description

Begin a series of plots using subtables constructed via an iteration axes. In addition to plotdata parameters, set a list of iteration axes (Table column names) and use iterplotnext() to step through. Only forward step through is allowed.

Arguments

Inputs	
poption	Record of plot options default is nxpanels=1, nypanels=1, windowsize=6, aspectratio=0.8, fontsize=1.0 allowed: record Default: 1 1 6 0.8 1.0
labels	List of strings : Title,Xlabel,Ylabel allowed: stringArray Default:
datastr	List of TaQL strings : X,Y allowed: stringArray Default:
iteraxes	List of strings : Iteration axes allowed: stringArray Default:

Returns
bool

Example

```
# Open a list of MS tables to plot from,
```

```
# and initialize a plot of Amplitude vs UV distance for
# channel 1 and stokes 1, iterating over Antenna1
tp.open(tabnames=['3c273.ms','3c48.ms'])
pop = { 'nrows':3, 'ncols':1,'panel':1, 'plotcolour':1,
        'aspectratio':1.6}
iteraxes = ['ANTENNA1']
labels = ['Amplitude vs UVdist','uvdist','amplitude']
xystr = ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1,1:2])']
tp.iterplotstart(poption=pop, labels=labels, datastr=xystr,
                 iteraxes=iteraxes)
tp.iterplotnext()
```

Example

To iterate over baseline and plot Amplitude vs time, for stokes 1, channel 1.

```
pop = { 'nrows':4, 'ncols':1 }
labels = ['Amplitude vs UVdist (iterating over Baseline)',
           'uvdist','amplitude']
xystr = ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1,1])']
iteraxes = ['ANTENNA1','ANTENNA2']
tp.iterplotstart(poption=plotopts,labels=labels,
                 datastr=xystr,iteraxes=iteraxes)
```

---

tableplot.replot.html

tableplot.replot - Function

3.2.2 Replot all existing panels and layers.

Description

Replot all existing panels and layers. To be used after a change of flag version, to get all visible plots to reflect the changed flags.

Arguments

Inputs

Returns

bool

---

tableplot.iterplotnext.html

tableplot.iterplotnext - Function

3.2.2 Start/Continue plotting

Description

Start/Continue plotting by stepping through the iteration axes. Call after tp.iterplotstart(). Returns 1 if additional iteration steps remain , 0 if last iteration has completed.

Arguments

Returns
int

Example

```
# iterate through the data

tp.open(tabnames=['3c273.ms','3c48.ms'])
plotopts = {'aspectratio': 1.2, 'ncols': 2, 'nrows': 1}
labels = ['Amplitude vs UVdist (iterating over Antenna1)',
          'uvdist','amplitude']
xystr = ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1,1])']
iteraxes = ['ANTENNA1']
tp.iterplotstart(poption=plotopts,labels=labels,
                 datastr=xystr,iteraxes=iteraxes)
ret = tp.iterplotnext()
ret = tp.iterplotnext()
ret = tp.iterplotnext()
...
```

tableplot.iterplotstop.html

tableplot.iterplotstop - Function

3.2.2 Stop plot iterations.

Description

To be called at the end of the plot iterations, or in between if desired. Okay if ignored.

Arguments

Inputs	
rmplotter	Indicates whether the plot window should be removed (true) from the display or left (false) allowed: bool Default: false

Returns

bool

Example

```
# iterate through and stop after 5 iterations of 2 plots per page

tp.open(tabnames=['3c273.ms','3c48.ms'])
plotopts = {'ncols': 2, 'nrows':1 }
labels = ['Amplitude vs UVdist (iterating over Antenna1)',
          'uvdist','amplitude']
xystr = ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1,1])']
iteraxes = ['ANTENNA1']
tp.iterplotstart(poption=plotopts,labels=labels,
                 datastr=xystr,iteraxes=iteraxes)
ret = tp.iterplotnext()
ret = tp.iterplotnext()
ret = tp.iterplotnext()
```

```
ret = tp.iterplotnext()
ret = tp.iterplotnext()
tp.iterplotstop()
```

---

tableplot.markregions.html

tableplot.markregions - Function

3.2.2 Mark a rectangular region to flag

Description

Mark or specify a rectangular region to flag. Call without arguments to enable mouse based interactive region marking. Marked regions can be discarded via the 'Alt' key. Command-line region marking can be done by setting panel and region parameters. After marking flag regions, call tp.flagdata() or tp.unflagdata().

Arguments

Inputs		
nrows	Number of rows of panels	
	allowed:	int
	Default:	0
ncols	Number of columns of panels	
	allowed:	int
	Default:	0
panel	Panel number	
	allowed:	int
	Default:	1
region	[xmin,ymin,xmax,ymax] bounding box	
	allowed:	doubleArray
	Default:	0.0

Returns
bool

Example

```
tp.markregions(nrows=2,ncols=1,panel=1,region=[300.0,400.0,0.090,0.095])
```

tableplot.flagdata.html

tableplot.flagdata - Function

3.2.2 Flag Data for selected flag regions

Description

Set flags for all regions marked using `tp.markregions()`. The plot is automatically redrawn after applying flags.
If reduction TaQL functions such as `sum,mean` are used, flags corresponding to all accessed values will be modified. For example, with a measurement set table, flagging on the mean amplitude of stokes 1 and channels 1 to 5, given by 'MEAN(AMPLITUDE(DATA[1,1:5]))' results in flags being set for all 5 accessed channels.
For a measurement set, by default, flags are set only for accessed channels and stokes when the DATA column is used. However all channels/stokes can be flagged for the marked flag regions by setting the corresponding row flag.

Arguments

Inputs

Returns

bool

---

tableplot.unflagdata.html

tableplot.unflagdata - Function

3.2.2 Unset flags for all regions marked using `tp.markregions()`.

Description

Unset flags for all regions marked using `tp.markregions()`. This is similar to the tp.flagdata() function in all other respects.

Arguments

Inputs

Returns

bool

Example

```
# mark 2 flag regions on a multi-panel plot, one in panel 1 and one
# in panel 2. Then apply the flags and write to disk.
  tp.markregions(panel=1)
  tp.markregions(panel=2)
  tp.unflagdata()
```

---

tableplot.locatedata.html

tableplot.locatedata - Function

3.2.2 Print info about data selected using tp.markregions().

Description

Print info about data selected using tp.markregions().

Arguments

Inputs		
columnlist	List of strings :	Column names (or TaQL expressions !)
	allowed:	stringArray
	Default:	

Returns

bool

---

tableplot.clearflags.html

tableplot.clearflags - Function

3.2.2 Clear all flags in the table.

Description

Clear all flags from the table. This may eventually be modified to allow for selective un-flagging of previously flagged regions (specified by indexing into a stored history of marked flag-regions).

Arguments

Inputs	
roottable	false : clear flags for the current sub-selection; true : clear flags for root table
	allowed: bool
	Default: false

Returns

bool

Example

```
# clear all flags from two measurement set tables

  tp.open(tabnames=['3c273.ms','3c48.ms'])
  tp.clearflags()
  tp.done()
```

tableplot.saveflagversion.html

tableplot.saveflagversion - Function

3.2.2 Save current flags with a version name.

Description

Save current flags with a version name. This applies to the last opened Tables

Arguments

Inputs		
versionname	Version name	
	allowed:	string
	Default:	
comment	Comment for this flag table	
	allowed:	string
	Default:	
merge	merge type	
	allowed:	string
	Default:	

Returns

bool

---

tableplot.restoreflagversion.html

tableplot.restoreflagversion - Function

3.2.2 Restore flags from a saved flag_version.

Description

Restore flags from a saved flag_version. This applies to the last opened Tables versionname : name of flag version to restore to main table merge : Type of operation to perform during restoration. merge = replace : replaces the main table flags. merge = and : logical AND with main table flags merge = or : logical OR with main table flags Default : replace.

Arguments

Inputs		
versionname	Version name	
	allowed:	stringArray
	Default:	
merge	merge type	
	allowed:	string
	Default:	

Returns

bool

---

tableplot.deleteflagversion.html

tableplot.deleteflagversion - Function

3.2.2 Delete a saved flag_version.

Description

Delete a saved flag_version. This applies to the last opened Tables

Arguments

Inputs		
versionname	Version name	
	allowed:	stringArray
	Default:	

Returns

bool

---

tableplot.getflagversionlist.html

tableplot.getflagversionlist - Function

3.2.2 Print out a list of saved flag_versions.

Description

Print out a list of saved flag_versions. This applies to the last opened Tables

Arguments

Inputs

Returns

bool

---

tableplot.clearplot.html

tableplot.clearplot - Function

3.2.2 Clear a plot.

Description

Clear a plot. An empty argument list (i.e., clearplot()) or clearplot(0) clears all plots currently visible whereas clearplot(nrows,ncols,panel) clears a plot on a specified panel.

Arguments

Inputs		
nrows	Number of rows of panels	
	allowed:	int
	Default:	0
ncols	Number of columns of panels	
	allowed:	int
	Default:	0
panel	Panel number (index)	
	allowed:	int
	Default:	0

Returns

bool

Example

```
# clear all flags from two measurement set tables

  tp.open(tabnames=['3c273.ms','3c48.ms'])
  tp.clearflags()
  tp.done()
```

tableplot.done.html

tableplot.done - Function

3.2.2 Clean up the tableplot tool

Description

Clean up the tableplot tool, and make it ready for tp.open() again.

Arguments

Returns
bool

ThirdParty.html

Chapter 4

Package ThirdParty

The ThirdParty package contains modules that interface to 3rd party software. atmosphere-Module.html

4.1 atmosphere - Module

Module for interfacing to Juan R. Padro's ATM library

Description This is a thin-interface to Juan R. Padro's proprietary ATM library atmlib. The atm library can be used to calculate atmospheric opacity and phase, and to perform radiative transfer calculations. In CASA, it is used for calibration, phase correction and for simulations. See Juan Padro's web page for futher information about the library structure and for information about other uses.

atmosphere-Tool.html

4.1.1 atmosphere - Tool

Atmosphere model
Requires:

Synopsis

Methods

atmosphere	Construct an atmosphere tool
getAtmVersion	Returns the version of ATM library.
listAtmosphereTypes	Returns a list of atmospheric types used by ATM.
initAtmProfile	Set initial atmospheric profile for atmosphere tool
updateAtmProfile	Update basic atmospheric parameters of atmosphere tool
getBasicAtmParms	Gets the current basic atmospheric parameters of the model.
getNumLayers	Returns the number of layers in the atmospheric profile.
getGroundWH2O	get the zenith column of water vapor
getProfile	get atmospheric profile
initSpectralWindow	initialize spectral window
addSpectralWindow	add a new spectral window
getNumSpectralWindows	Get number of spectral windows
getNumChan	return the number of channels of ith band
getRefChan	Get the reference channel of a given spectral window
getRefFreq	Get the reference frequency of given spectral window
getChanSep	Get the channel separation for regularly spaced grid for spectral window
getChanFreq	Get the channel frequency for a given grid point for the specified spectra
getSpectralWindow	Get the spectral grid for the specified spectral window.
getChanNum	Get the grid position for a given frequency in the specified spectral wind
getBandwidth	Get the frequency range encompassing the list of frequency grid points f
getMinFreq	Get lowest frequency channel for the specified spectral window.
getMaxFreq	Get highest frequency channel for the specified spectral window.
getDryOpacity	get the integrated Dry Opacity along the atmospheric path for channel r
getDryContOpacity	get the integrated Dry Continuum Opacity along the atmospheric path f
getO2LinesOpacity	get the integrated O2 Lines Opacity along the atmospheric path for chan
getO3LinesOpacity	get the integrated O3 Lines Opacity along the atmospheric path for chan
getCOLinesOpacity	get the integrated CO Lines Opacity along the atmospheric path for cha
getN2OLinesOpacity	get the integrated N2O Lines Opacity along the atmospheric path for ch
getWetOpacity	get the integrated Wet Opacity along the atmospheric path for channel r
getH2OLinesOpacity	get the integrated H2O Lines Opacity along the atmospheric path for ch
getH2OContOpacity	get the integrated H2O Continuum Opacity along the atmospheric path
getDryOpacitySpec	get the integrated Dry optical depth along the atmospheric path on each
getWetOpacitySpec	get the integrated Wet optical depth along the atmospheric path on each
getDispersivePhaseDelay	get the integrated zenith H2O Atmospheric Phase Delay
getDispersiveWetPhaseDelay	get the integrated dispersive wet Atmospheric Phase Delay

getNonDispersiveWetPhaseDelay	get the integrated nondispersive wet Atmospheric Phase Delay
getNonDispersiveDryPhaseDelay	get the integrated nondispersive dry Atmospheric Phase Delay
getNonDispersivePhaseDelay	get the integrated zenith H2O Atmospheric Phase Delay (Non-Dispersiv
getDispersivePathLength	get the integrated Atmospheric Dispersive Path
getDispersiveWetPathLength	get the integrated wet Atmospheric Dispersive Path
getNonDispersiveWetPathLength	get the integrated wet Atmospheric NonDispersive Path
getNonDispersiveDryPathLength	get the integrated dry Atmospheric NonDispersive Path
getO2LinesPathLength	get the integrated O2 lines Path
getO3LinesPathLength	get the integrated O3 lines Path
getCOLinesPathLength	get the integrated CO lines Path
getN2OLinesPathLength	get the integrated N2O lines Path
getNonDispersivePathLength	get the integrated Atmospheric Non-Dispersive Path
getAbsH2OLines	Get H2O lines Absorption Coefficient at layer nl and frequency channel
getAbsH2OCont	Get H2O continuum Absorption Coefficient at layer nl and frequency ch
getAbsO2Lines	Get O2 lines Absorption Coefficient at layer nl and frequency channel nf
getAbsDryCont	Get Dry Continuum Absorption Coefficient at layer nl and frequency cha
getAbsO3Lines	Get O3 lines Absorption Coefficient at layer nl and frequency channel nf
getAbsCOLines	Get CO lines Absorption Coefficient at layer nl and frequency channel n
getAbsN2OLines	Get N2O lines Absorption Coefficient at layer nl and frequency channel
getAbsTotalDry	Get Total Dry Absorption Coefficient at layer nl and frequency channel
getAbsTotalWet	Get total wet absorption coefficient at layer nl and frequency channel nf
setUserWH2O	set the user zenith water vapor column
getUserWH2O	get the user zenith water vapor column
setAirMass	Set the air mass
getAirMass	Get the air mass
setSkyBackgroundTemperature	Set the sky background temperature
getSkyBackgroundTemperature	Get the sky background temperature
getAverageTebbSky	Returns average equiv. BB Temp
getTebbSky	Returns equiv. BB Temp
getTebbSkySpec	Returns equiv. BB Temp on each channel of a band
getAverageTrjSky	Returns the average Rayleigh-Jeans Temperature
getTrjSky	Returns the Rayleigh-Jeans Temperature
getTrjSkySpec	Returns the Rayleigh-Jeans Temperatures on each channel of a band

atmosphere.atmosphere.html

atmosphere.atmosphere - Function

4.1.1 Construct an atmosphere tool

Description

This is used to construct an `atmosphere` tool.

Arguments

Inputs

Returns
atmosphere

Example

```
A default atmosphere tool is created automatically during casapy
startup and defined as 'at'.
Manual tool construction is done this way:

at = casac.atmosphere()
```

---

atmosphere.getAtmVersion.html

atmosphere.getAtmVersion - Function

4.1.1 Returns the version of ATM library.

Description

Returns the version of ATM library implemented to this tool.

Arguments

Returns
string

Example

```
at.getAtmVersion()
# 'ATM-0_5_0'
```

---

atmosphere.listAtmosphereTypes.html

atmosphere.listAtmosphereTypes - Function

4.1.1 Returns a list of atmospheric types used by ATM.

Description

Returns a list of index numbers and corresponding atmosphere types used by the ATM library.

Arguments

Returns
stringArray

Example

```
at.listAtmosphereTypes()
# ['1 - TROPICAL', '2 - MIDLATSUMMER', '3 - MIDLATWINTER',
#  '4 - SUBARTSUMMER', '5 - SUBARTWINTER']
```

---

atmosphere.initAtmProfile.html

atmosphere.initAtmProfile - Function

4.1.1 Set initial atmospheric profile for atmosphere tool

Description

An atmospheric profile is composed of 4 quantities as a function of altitude z: * the layer thickness * the pressure P * the temperature T and * the gas densities for H2O, O3, CO and N2O.
This method is needed for computing the absorption and phase coefficients, as well as for performing radiative transfer calculations (only layer thickness/T are needed).
This method builds an atmospheric profile that can be used to calculate absorption and phase coefficients, as well as to perform forward and/or retrieval radiative transfer calculations. It is composed of a set of parameters needed to build a layer thickness/P/T/gas densities densities profile from simple parameters currently available at observatories (from weather stations for example) using functions from the ATM library. The set of input parameters consists of the pressure P, the temperature T and the relative humidity at the ground, the altitude of the site, the tropospheric temperature lapse rate,... The profile is built as: thickness of the considered atmospheric layers above the site, and mean P,T,H2O,O3,CO,N2O in them. The total number of atmospheric layers in the particular profile is also available (a negative value indicates an error). The zenith column of water vapor can be calculated by simply integrating the H2O profile.

Arguments

Inputs		
altitude	m	Site altitude - Quantity with units of altitude, meter
		allowed: doublem
		Default: 5000.
temperature	K	Ambient Temperature - Quantity with units of temperature, K
		allowed: doubleK
		Default: 270.0
pressure	mbar	Ambient pressure - Quantity with units of pressure, mbar
		allowed: doublembar
		Default: 560.0
maxAltitude	km	altitude of the top pf the modelled atmosphere - Quantity with dimension of length, and units of kilometer
		allowed: doublekm
		Default: 48.0
humidity		used to guess water (0-100)
		allowed: double
		Default: 20.0
dTem_dh	K/km	the derivative of temperature with respect to height - Quantity with units of K/km
		allowed: doubleK/km
		Default: -5.6
dP	mbar	initial pressure step - Quantity with the units of pressure, mb
		allowed: doublembar
		Default: 10.0
dPm		pressure multiplicative factor for steps
		allowed: double
		Default: 1.2
h0	km	scale height for water(exp distribution) - Quantity with the dimension of length, and units of kilometer
		allowed: doublekm
		Default: 2.0
atmType		atmospheric type 1(tropical),2(mid latitude summer),3(mid latitude winter), 4(subarctic summer),5(subarctic winter), dimensionless
		allowed: int
		Default: 1

Returns
string

Example

```
tmp = qa.quantity(270.0, 'K')
pre = qa.quantity(560.0, 'mbar')
hum = 20.0
alt = qa.quantity(5000, 'm')
h0  = qa.quantity(2.0, 'km')
wvl = qa.quantity(-5.6, 'K/km')
mxA = qa.quantity(48, 'km')
dpr = qa.quantity(10.0, 'mbar')
dpm = 1.2
att = 1
myatm = at.initAtmProfile(alt, tmp, pre, mxA, hum, wvl, dpr, dpm, h0, att)
print myatm
# BASIC ATMOSPHERIC PARAMETERS TO GENERATE REFERENCE ATMOSPHERIC PROFILE
#
# Ground temperature T:         270 K
# Ground pressure P:            560 mb
# Relative humidity rh:         20 %
# Scale height h0:              2 km
# Pressure step dp:             10 mb
# Altitude alti:                5000 m
# Attitude top atm profile:     48 km
# Pressure step factor:         1.2
# Tropospheric lapse rate:      -5.6 K/km
# Atmospheric type:             TROPICAL
#
# Built atmospheric profile with 20 layers.
```

---

atmosphere.updateAtmProfile.html

atmosphere.updateAtmProfile - Function

4.1.1 Update basic atmospheric parameters of atmosphere tool

Description

This is used to update the `atmosphere` tool when basic atmospheric parameters.change.

Arguments

Inputs		
altitude	m	Site altitude - Quantity with units of altitude, meter
		allowed: doublem
		Default: 5000.
temperature	K	Ambient ground temperature - Quantity with units of temperature, K
		allowed: doubleK
		Default: 270.0
pressure	mbar	Ambient ground pressure - Quantity with units of pressure, mbar
		allowed: doublembar
		Default: 560.0
humidity		Relative humidy used to guess water (0-100)
		allowed: double
		Default: 20.0
dTem_dh	K/km	Tropospheric Lapse Rate - the derivative of temperature with respect to height - Quantity with units of K/km
		allowed: doubleK/km
		Default: -5.6
h0	km	scale height for water(exp distribution) - Quantity with the dimension of length, and units of kilometer
		allowed: doublekm
		Default: 2.0

Returns
string

Example

```
new_tmp = qa.quantity(275.0, 'K')
print at.updateAtmProfile(alt, new_tmp, pre, hum, wvl, h0)
# UPDATED BASIC ATMOSPHERIC PARAMETERS TO GENERATE REFERENCE ATMOSPHERIC PROFILE
#
# Ground temperature T:         275 K
# Ground pressure P:            560 mb
# Relative humidity rh:         20 %
# Scale height h0:              2 km
# Altitude alti:                5000 m
# Tropospheric lapse rate:      -5.6 K/km
```

---

atmosphere.getBasicAtmParms.html

atmosphere.getBasicAtmParms - Function

4.1.1 Gets the current basic atmospheric parameters of the model.

Arguments

Outputs		
altitude	m	Site altitude - Quantity with units of altitude, meter allowed: doublem Default:
temperature	K	Ambient ground temperature - Quantity with units of temperature, K allowed: doubleK Default:
pressure	mbar	Ambient pressure - Quantity with units of pressure, mbar allowed: doublembar Default:
maxAltitude	km	altitude of the top pf the modelled atmosphere - Quantity with dimension of length, and units of kilometer allowed: doublekm Default:
humidity		Ground relative humidity used to guess water (0-100) allowed: double Default:
dTem_dh	K/km	Current Tropospheric Lapse Rate (the derivative of temperature with respect to height) - Quantity with units of K/km allowed: doubleK/km Default:
dP	mbar	initial pressure step - Quantity with the units of pressure, mb allowed: doublembar Default:
dPm		pressure multiplicative factor for steps allowed: double Default:
h0	km	Water vapor scale height (exp distribution) - Quantity with the dimension of length, and units of kilometer allowed: doublekm Default:
atmType		atmospheric type used to describe the behaviour above the tropopause. 1(tropical),2(mid latitude summer),3(mid latitude winter), 4(subarctic summer),5(subarctic winter), dimensionless allowed: string Default:

Returns
string

Example

```
p = at.getBasicAtmParms()
# returns a tuple of
# 0 - string listing of parameters, 1 - altitude, 2 - temperature,
# 3 - pressure, 4 - maxAltitude, 5 - humidity, 6 - dTem_dh,
# 7 - dP, 8 - dPm. 9 - h0, and 10 - atmType
print "Atmospheric type:   ", p[10]
# Atmospheric type:    TROPICAL
print "Ground temperature: ", p[2]['value'][0], p[2]['unit']
# Ground temperature:  288.16 K
print p[0]           # a "pretty" listing of all the parameters
# CURRENT ATMOSPHERIC PARAMETERS OF REFERENCE ATMOSPHERIC PROFILE
#
# Ground temperature T:         275 K
# Ground pressure P:            560 mbar
# Relative humidity rh:         20 %
# Scale height h0:              2 km
# Pressure step dp:             10 mbar
# Altitude alti:                5000 m
# Attitude top atm profile      48 km
# Pressure step factor          1.2
# Tropospheric lapse rate       -5.6 K/km
# Atmospheric type:             TROPICAL
#
# Atmospheric profile has 20 layers.
```

---

atmosphere.getNumLayers.html

atmosphere.getNumLayers - Function

4.1.1 Returns the number of layers in the atmospheric profile.

Arguments

Returns
int

Example

```
p = at.getProfile()
for i in range(at.getNumLayers()):
        # Print atmospheric profile returned by at.getProfile():
        # Layer thickness (idx=1), Temperature (idx=2),
        # Number density of water vapor(idx=4), and Pressure (idx=5)
        print p[1]['value'][i], p[2]['value'][i], p[4]['value'][i], p[5]['value'][i]
```

---

atmosphere.getGroundWH2O.html

atmosphere.getGroundWH2O - Function

4.1.1 get the zenith column of water vapor

Description

Method to get the zenith column of water vapor. It is computed by simply integrating the H2O profile:

Arguments

Returns
Quantity

Example

```
w = at.getGroundWH2O()
print "Guessed water content: ", w['value'][0], w['unit']
# Guessed water content:  2.6529103462750112 mm
```

---

atmosphere.getProfile.html

atmosphere.getProfile - Function

4.1.1 get atmospheric profile

Description

Get the atmospheric profile.

Arguments

Outputs		
thickness	m	thickness of every atmospheric layer - Quantum with a vector value and unit of length, m allowed: doublem Default:
temperature	K	temperature of every atmospheric layer - Quantum with a vector value and unit of temperature, K allowed: doubleK Default:
watermassdensity	kg.m-3	water vapor mass density content of every atmospheric layer - Quantum with a vector value and unit of kg.m-3 allowed: doublekg.m-3 Default:
water	m-3	water vapor content of every atmospheric layer - Quantum with a vector value and unit of m-3 allowed: doublem-3 Default:
pressure	Pa	pressure of every atmospheric layer - Quantum with a vector value and unit of Pascal allowed: doublePa Default:
O3	m-3	O3 of every atmospheric layer - Quantum with a vector value and unit of m-3 allowed: doublem-3 Default:
CO	m-3	CO of every atmospheric layer - Quantum with a vector value and unit of m-3 allowed: doublem-3 Default:
N2O	m-3	N2O of every atmospheric layer - Quantum with a vector value and unit of m-3 allowed: doublem-3 Default:

Returns
string

Example

```
p = at.getProfile()
```

```
# returns a tuple of
# 0 - string listing of layer values, and arrays of layer, 1 - thickness,
# 2 - temperature, 3 - watermassdensity, 4 - water (number density),
# 5 - pressure, 6 - O3 (number density), 7 - CO, 8 - N2O
for i in range(at.getNumLayers()):
        # Print atmospheric profile returned by at.getProfile():
        # Layer thickness (idx=1), Temperature (idx=2),
        # Number density of water vapor(idx=4), and Pressure (idx=5)
        print p[1]['value'][i], p[2]['value'][i], p[4]['value'][i],\
              p[5]['value'][i]

print p[0]    # "pretty" listing of all layer parameters
```

---

atmosphere.initSpectralWindow.html

atmosphere.initSpectralWindow - Function

4.1.1 initialize spectral window

Description

function that defines a spectral window, computes absorption and emmision coefficients for this window, using the above atmospheric parameters.

Arguments

Inputs		
nbands		number of spectral windows/bands allowed: int Default: 1
fCenter	GHz	center frequencies - Quantum with a vector value and unit of frequency, GHz allowed: doubleGHz Default: 90
fWidth	GHz	frequency width of band - Quantum with a vector value and unit of frequency, GHz allowed: doubleGHz Default: 0.64
fRes	GHz	resolution inside band - Quantum with a vector value and unit frequency, GHz. Default is for a single frequency. allowed: doubleGHz Default: 0.0

Returns

int

Example

```
nb = 1
```

```
fC = qa.quantity(88., 'GHz')
fW = qa.quantity(0.5, 'GHz')
fR = qa.quantity(0.5, 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)

nb = 2
fC = qa.quantity([88., 90.], 'GHz')
fW = qa.quantity([0.5, 0.5], 'GHz')
fR = qa.quantity([0.125, 0.125], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
```

---

atmosphere.addSpectralWindow.html

atmosphere.addSpectralWindow - Function

4.1.1 add a new spectral window

Description

Add a new spectral window, uniformly sampled, this spectral window having no sideband.

Arguments

Inputs		
fCenter	GHz	frequencies - Quantum with a double value and unit of frequency, GHz allowed: doubleGHz Default: 350
fWidth	GHz	frequency width of band - Quantum with a double value and unit of frequency, GHz allowed: doubleGHz Default: 0.008
fRes	GHz	resolution inside band - Quantum with a double value and unit frequency, GHz allowed: doubleGHz Default: 0.002

Returns

int

Example

```
fC2 = qa.quantity(350.0, 'GHz')
fW2 = qa.quantity(0.008, 'GHz')
fR2 = qa.quantity(0.002, 'GHz')
nc = at.addSpectralWindow(fC2, fW2, fR2)
print "New spectral window has ", nc, " channels"
```

atmosphere.getNumSpectralWindows.html

atmosphere.getNumSpectralWindows - Function

4.1.1 Get number of spectral windows

Arguments

Returns
int

Example

```
numSpw = at.getNumSpectralWindows()
print "There are ", numSpw, " spectral windows"
```

---

atmosphere.getNumChan.html

atmosphere.getNumChan - Function

4.1.1 return the number of channels of ith band

Description

Return the number of channels of ith band (passes in as parameter).

Arguments

Inputs		
spwid	Int standing for identifier of bands (0-based)	
	allowed:	int
	Default:	0

Returns
int

Example

```
for spwid in range(at.getNumSpectralWindows()):
        numCh = at.getNumChan(spwid)
        print "Spectral window ", spwid, " has ", numCh, " frequency channels"
```

---

atmosphere.getRefChan.html

atmosphere.getRefChan - Function

4.1.1 Get the reference channel of a given spectral window

Description

Return the reference channel of the given spectral window

Arguments

Inputs		
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
int

Example

```
rc = at.getRefChan()
print "Reference channel retrieved: ", rc
```

---

atmosphere.getRefFreq.html

atmosphere.getRefFreq - Function

4.1.1 Get the reference frequency of given spectral window

Description

Return the reference frequency of the given spectral window

Arguments

Inputs		
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
Quantity

Example

```
rf = at.getRefFreq()
print "Reference frequency retrieved: ", rf['value'][0], rf['unit']
```

---

atmosphere.getChanSep.html

atmosphere.getChanSep - Function

4.1.1 Get the channel separation for regularly spaced grid for spectral window

Description

Return the channel separation of the given spectral window

Arguments

Inputs		
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
Quantity

Example

```
cs = at.getChanSep()
print "Channel separation retrieved: ", cs['value'][0], cs['unit']
```

---

atmosphere.getChanFreq.html

atmosphere.getChanFreq - Function

4.1.1 Get the channel frequency for a given grid point for the specified spectral window.

Description

Return the channel frequency for a given grid point for the specified spectral window.

Arguments

Inputs		
chanNum	Int standing for channel number (0-based)	
	allowed:	int
	Default:	0
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
Quantity

Example

```
for spwid in range(at.getNumSpectralWindows()):
        numCh = at.getNumChan(spwid)
        print "Spectral window ", spwid, " has ", numCh, " frequency channels"
        for n in range(numCh):
                freq = at.getChanFreq(n, spwid)
                print "Channel ", n, " Frequency:", freq['value'][0], freq['unit']
```

atmosphere.getSpectralWindow.html

atmosphere.getSpectralWindow - Function

4.1.1 Get the spectral grid for the specified spectral window.

Description

Return the spectral grid for the specified spectral window.

Arguments

Inputs		
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
print at.getSpectralWindow()['value'],at.getSpectralWindow()['unit']

sg = at.getSpectralWindow()
for i in range(len(sg['value'])):
        print sg['value'][i], sg['unit']
```

---

atmosphere.getChanNum.html

atmosphere.getChanNum - Function

4.1.1 Get the grid position for a given frequency in the specified spectral window.

Description

Return the channel number for given frequency in the specified spectral window relative to the reference channel number.

Arguments

Inputs			
freq	GHz	Frequency	
		allowed:	doubleGHz
		Default:	
spwid		Int standing for spectral window id (0-based)	
		allowed:	int
		Default:	0

Returns
double

Example

```
# List current spectral window setting of SPW0
at.getRefFreq()['value'][0], at.getRefFreq()['unit']
# (90.0, 'GHz')
print at.getChanSep()['value'][0], at.getChanSep()['unit']
# 10.0 MHz
at.getRefChan()
# 32

# Get grid positions
at.getChanNum(qa.quantity(90., 'GHz'))
```

```
# 0.0

at.getChanNum(qa.quantity(90., 'GHz'), 0)
# 0.0

at.getChanNum(qa.quantity(90.08, 'GHz'), 0)
# 8.0

at.getChanNum(qa.quantity(89.985, 'GHz'), 0)
# -1.5

at.getChanNum(qa.quantity(89.98,'GHz'), 0)
# -2.0
```

---

atmosphere.getBandwidth.html

atmosphere.getBandwidth - Function

4.1.1 Get the frequency range encompassing the list of frequency grid points for the specified spectral window.

Description

Get the frequency range encompassing the list of frequency grid points for the specified spectral window.

Arguments

Inputs		
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
Quantity

Example

```
print "Total bandwidth retrieved: ", at.getBandwidth()['value'][0], at.getBandwidth()['unit'
```

atmosphere.getMinFreq.html

atmosphere.getMinFreq - Function

4.1.1 Get lowest frequency channel for the specified spectral window.

Description

Get lowest frequency channel for the specified spectral window.

Arguments

Inputs		
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
print "Frequency range: from ", at.getMinFreq()['value'][0], " to ",\
      at.getMaxFreq()['value'][0], at.getMinFreq()['unit']
```

atmosphere.getMaxFreq.html

atmosphere.getMaxFreq - Function

4.1.1 Get highest frequency channel for the specified spectral window.

Description

Get highest frequency channel for the specified spectral window.

Arguments

Inputs		
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
print "Frequency range: from ", at.getMinFreq()['value'][0], " to ",\
      at.getMaxFreq()['value'][0], at.getMaxFreq()['unit']
```

---

atmosphere.getDryOpacity.html

atmosphere.getDryOpacity - Function

4.1.1 get the integrated Dry Opacity along the atmospheric path for channel nc in spectral window swpId

Description

Get the integrated Dry Opacity for one channel in a band.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
double

Example

```
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total Dry Opacity at ", fC['value'][0], fC['unit'],\
      " for 1.0 air mass: ", at.getDryOpacity()
```

atmosphere.getDryContOpacity.html

atmosphere.getDryContOpacity - Function

4.1.1 get the integrated Dry Continuum Opacity along the atmospheric path for channel nc in spectral window spwid

Description

Get the integrated Dry Continuum Opacity for one channel in a band.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
double

Example

```
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total Dry Cont Opacity at ", fC['value'][0], fC['unit'],\
      " for 1.0 air mass: ", at.getDryContOpacity()
```

atmosphere.getO2LinesOpacity.html

atmosphere.getO2LinesOpacity - Function

4.1.1 get the integrated O2 Lines Opacity along the atmospheric path for channel nc in spectral window spwid

Description

Get the integrated O2 Lines Opacity for one channel in a band.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
double

Example

```
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total O2 Lines Opacity at ", fC['value'][0], fC['unit'],\
      " for 1.0 air mass: ", at.getO2LinesOpacity()
```

atmosphere.getO3LinesOpacity.html

atmosphere.getO3LinesOpacity - Function

4.1.1 get the integrated O3 Lines Opacity along the atmospheric path for channel nc in spectral window spwid

Description

Get the integrated O3 Lines Opacity for one channel in a band.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
double

Example

```
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total O3 Lines Opacity at ", fC['value'][0], fC['unit'],\
      " for 1.0 air mass: ", at.getO3LinesOpacity()
```

atmosphere.getCOLinesOpacity.html

atmosphere.getCOLinesOpacity - Function

4.1.1 get the integrated CO Lines Opacity along the atmospheric path for channel nc in spectral window spwid

Description

Get the integrated CO Lines Opacity for one channel in a band.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
double

Example

```
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total CO Lines Opacity at ", fC['value'][0], fC['unit'],\
      " for 1.0 air mass: ", at.getCOLinesOpacity()
```

atmosphere.getN2OLinesOpacity.html

atmosphere.getN2OLinesOpacity - Function

4.1.1 get the integrated N2O Lines Opacity along the atmospheric path for channel nc in spectral window spwid

Description

Get the integrated N2O Lines Opacity for one channel in a band.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
double

Example

```
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total N2O Lines Opacity at ", fC['value'][0], fC['unit'],\
      " for 1.0 air mass: ", at.getN2OLinesOpacity()
```

atmosphere.getWetOpacity.html

atmosphere.getWetOpacity - Function

4.1.1 get the integrated Wet Opacity along the atmospheric path for channel nc in spectral window spwid

Description

Get the integrated Wet Opacity for one channel in a band.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
Quantity

Example

```
for i in range(at.getNumSpectralWindows()):
        for j in range(at.getNumChan(i)):
                print "Frequency: ", at.getChanFreq(j, i)['value'][0], at.getChanFreq(j, i)
                print "Wet opacity:", at.getWetOpacity(j, i)['value'][0], at.getWetOpacity(j
                      " for ", at.getUserWH2O()['value'][0], at.getUserWH2O()['unit'], " H2O
```

atmosphere.getH2OLinesOpacity.html

atmosphere.getH2OLinesOpacity - Function

4.1.1 get the integrated H2O Lines Opacity along the atmospheric path for channel nc in spectral window spwid

Description

Get the integrated H2O Lines Opacity for one channel in a band.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns

double

Example

```
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total H2O Lines Opacity at ", fC['value'][0], fC['unit'],\
      " for 1.0 air mass: ", at.getH2OLinesOpacity()
```

atmosphere.getH2OContOpacity.html

atmosphere.getH2OContOpacity - Function

4.1.1 get the integrated H2O Continuum Opacity along the atmospheric path for channel nc in spectral window spwid

Description

Get the integrated H2O Continuum Opacity for one channel in a band.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
double

Example

```
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total H2O Cont Opacity at ", fC['value'][0], fC['unit'],\
      " for 1.0 air mass: ", at.getH2OContOpacity()
```

atmosphere.getDryOpacitySpec.html

atmosphere.getDryOpacitySpec - Function

4.1.1 get the integrated Dry optical depth along the atmospheric path on each channel of a band

Description

Get the integrated Dry optical depth along the atmospheric path on each channel in a band.

Arguments

Outputs		
dryOpacity	dry opacity for each channel	
	allowed:	doubleArray
	Default:	
Inputs		
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns

int

Example

```
at.getDryOpacitySpec()
# (8,
#  array([0.12113794420465548, 0.11890122206854335,
#         0.11713584932434795, 0.11572780449702716,
#         0.11459567027114714, 0.11368004975916192,
#         0.11293678422232195,0.11233248854020933]))
```

atmosphere.getWetOpacitySpec.html

atmosphere.getWetOpacitySpec - Function

4.1.1 get the integrated Wet optical depth along the atmospheric path on each channel of a band

Description

Get the integrated optical Wet depth along the atmospheric path on each channel in a band.

Arguments

Outputs		
wetOpacity	mm-1	wet opacity for each channel in band - Quantum with a vector value and unit of mm-1 allowed: doublemm-1 Default:
Inputs		
spwid		Int standing for spectral window id (0-based) allowed: int Default: 0

Returns
int

Example

```
sw=at.getWetOpacitySpec()
# returns a tuple of
# 0 - The number of channels and
# 1 - an quantity array of wet opacity for each channel in band
sw[1]['value']
# array([1.7225454913767393, 1.7204246078103735,
#        1.7188614166349163, 1.7179243635081174,
#        1.7177278069990962, 1.7184525049248152,
```

```
#        1.7204244157129918, 1.7242351137518073])

sw[0]
# 8

Another example:
for s in range(at.getNumSpectralWindows()):
        print "band", s
        for i in range(at.getNumChan(0)):
                print " - dryOpacity ", at.getDryOpacitySpec(spwid=s)[1][i], " wet Opacity/m
                      at.getWetOpacitySpec(spwid=s)[1]['value'][i]
```

---

atmosphere.getDispersivePhaseDelay.html

atmosphere.getDispersivePhaseDelay - Function

4.1.1 get the integrated zenith H2O Atmospheric Phase Delay

Description

Get the integrated zenith H2O Atmospheric Phase Delay (Dispersive part) for the current conditions, for channel number nc of spectral window spwid.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
w = at.getUserWH2O()
numSpw = at.getNumSpectralWindows()
for spwid in range(numSpw):
        numCh = at.getNumChan(spwid)
        print "Spectral window ", spwid, " has ", numCh, " frequency channels"
        for n in range(numCh):
                freq = at.getChanFreq(n, spwid)
                print "Total Dispersive Phase Delay at ",freq['value'][0], freq['unit'], " f
                      (at.getDispersivePhaseDelay(n, spwid)['value'][0])/(w['value'][0])," c
                      ((100*at.getDispersivePhaseDelay(n, spwid)['value'][0])/(w['value'][0
                      "% of the Non-dispersive one )"
```

atmosphere.getDispersiveWetPhaseDelay.html

atmosphere.getDispersiveWetPhaseDelay - Function

4.1.1 get the integrated dispersive wet Atmospheric Phase Delay

Description

Function to retrieve the integrated wet Atmospheric Phase Delay (Dispersive part) along the atmospheric path corresponding to the 1st guess water column.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
Quantity

Example

```
w = at.getUserWH2O()
numSpw = at.getNumSpectralWindows()
for spwid in range(numSpw):
        numCh = at.getNumChan(spwid)
        print "Spectral window ", spwid, " has ", numCh, " frequency channels"
        for n in range(numCh):
                freq = at.getChanFreq(n, spwid)
                print "Total Dispersive Wet Phase Delay at ", freq['value'][0], freq['unit']
                      (at.getDispersiveWetPhaseDelay(n, spwid)['value'][0])/(w['value'][0]),
                      ((100*at.getDispersiveWetPhaseDelay(n, spwid)['value'][0])/(w['value']
                      "% of the Non-dispersive one )"
```

atmosphere.getNonDispersiveWetPhaseDelay.html

atmosphere.getNonDispersiveWetPhaseDelay - Function

4.1.1 get the integrated nondispersive wet Atmospheric Phase Delay

Description

Function to retrieve the integrated wet Atmospheric Phase Delay (NonDispersive part) along the atmospheric path corresponding to the 1st guess water column.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
Quantity

Example

```
w = at.getUserWH2O()
numSpw = at.getNumSpectralWindows()
for spwid in range(numSpw):
        numCh = at.getNumChan(spwid)
        print "Spectral window ", spwid, " has ", numCh, " frequency channels"
        for n in range(numCh):
                freq = at.getChanFreq(n, spwid)
                print "Total Dispersive Wet Phase Delay at ", freq['value'][0], freq['unit']
                      (at.getDispersiveWetPhaseDelay(n, spwid)['value'][0])/(w['value'][0]),
                      ((100*at.getDispersiveWetPhaseDelay(n, spwid)['value'][0])/(w['value']
```

```
"% of the Non-dispersive one )"
```

---

atmosphere.getNonDispersiveDryPhaseDelay.html

atmosphere.getNonDispersiveDryPhaseDelay - Function

4.1.1 get the integrated nondispersive dry Atmospheric Phase Delay

Description

Function to retrieve the integrated dry Atmospheric Phase Delay (NonDispersive part) along the atmospheric path corresponding to the 1st guess water column.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
w = at.getUserWH2O()
numSpw = at.getNumSpectralWindows()
for spwid in range(numSpw):
        numCh = at.getNumChan(spwid)
        print "Spectral window ", spwid, " has ", numCh, " frequency channels"
        for n in range(numCh):
                freq = at.getChanFreq(n, spwid)
                print "Total Dispersive Dry Phase Delay at ", freq['value'][0], freq['unit']
                      (at.getDispersiveDryPhaseDelay(n,spwid)['value'][0])/(w['value'][0]),'
                      ((100*at.getDispersiveDryPhaseDelay(n,spwid)['value'][0])/(w['value']
```

```
"% of the Non-dispersive one )"
```

---

atmosphere.getNonDispersivePhaseDelay.html

atmosphere.getNonDispersivePhaseDelay - Function

4.1.1 get the integrated zenith H2O Atmospheric Phase Delay (Non-Dispersive part)

Description

Get the integrated zenith H2O Atmospheric Phase Delay (Non-Dispersive part) for the current conditions, for channel number nc of spectral window spwid.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
w = at.getUserWH2O()
numSpw = at.getNumSpectralWindows()
for spwid in range(numSpw):
        numCh = at.getNumChan(spwid)
        print "Spectral window ", spwid, " has ", numCh, " frequency channels"
        for n in range(numCh):
                freq = at.getChanFreq(n, spwid)
                print "Total Dispersive Phase Delay at ", freq['value'][0], freq['unit'], "
                      (at.getDispersivePhaseDelay(n,spwid)['value'][0])/(w['value'][0])," de
```

```
((100*at.getDispersivePhaseDelay(n,spwid)['value'][0])/(w['value'][0])
"% of the Non-dispersive one )"
```

---

atmosphere.getDispersivePathLength.html

atmosphere.getDispersivePathLength - Function

4.1.1 get the integrated Atmospheric Dispersive Path

Description

Retrieve the integrated Atmospheric Path length (Dispersive part) along the atmospheric path corresponding to the user water column for channel nc in spectral window spwid.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
Quantity

Example

```
w = at.getUserWH2O()
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
nfR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total Dispersive Delay at ", fC['value'][0], fC['unit'], " for 1.0 air mass: ",\
      at.getDispersivePathLength()['value'][0] /  w['value'][0], " meters per mm of water va
print "(",100*(at.getDispersivePathLength()['value'][0] /  w['value'][0])/(at.getNonDispersi
      "% of the Non-dispersive one )"
```

atmosphere.getDispersiveWetPathLength.html

atmosphere.getDispersiveWetPathLength - Function

4.1.1 get the integrated wet Atmospheric Dispersive Path

Description

Retrieve the integrated wet Atmospheric Path length (Dispersive part) along the atmospheric path corresponding to the 1st guess water column for channel nc in spectral window spwid.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
w = at.getGroundWH2O()
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
nfR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total Dispersive Delay at ", fC['value'][0], fC['unit'], " for 1.0 air mass: ",\
      at.getDispersiveWetPathLength()['value'][0] /  w['value'][0], " meters per mm of wate
print "(",100*(at.getDispersiveWetPathLength()['value'][0] /  w['value'][0])/(at.getNonDispe
      "% of the Non-dispersive one )"
```

atmosphere.getNonDispersiveWetPathLength.html

atmosphere.getNonDispersiveWetPathLength - Function

4.1.1 get the integrated wet Atmospheric NonDispersive Path

Description

Retrieve the integrated wet Atmospheric Path length (NonDispersive part) along the atmospheric path corresponding to the 1st guess water column for channel nc in spectral window spwid.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
w = at.getGroundWH2O()
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
nfR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total Dispersive Delay at ", fC['value'][0], fC['unit'], " for 1.0 air mass: ",\
      at.getDispersiveWetPathLength()['value'][0] /  w['value'][0], " meters per mm of wate
print "(",100*(at.getDispersiveWetPathLength()['value'][0] /  w['value'][0])/(at.getNonDispe
      "% of the Non-dispersive one )"
```

atmosphere.getNonDispersiveDryPathLength.html

atmosphere.getNonDispersiveDryPathLength - Function

4.1.1 get the integrated dry Atmospheric NonDispersive Path

Description

Retrieve the integrated dry Atmospheric Path length (NonDispersive part) along the atmospheric path corresponding to the 1st guess water column for channel nc in spectral window spwid.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
Quantity

Example

```
w = at.getGroundWH2O()
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
nfR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total Dispersive Delay at ", fC['value'][0], fC['unit'], " for 1.0 air mass: ",\
      at.getDispersiveDryPathLength()['value'][0] /  w['value'][0], " meters per mm of wate
print "(",100*(at.getDispersiveDryPathLength()['value'][0] /  w['value'][0])/(at.getNonDispe
      "% of the Non-dispersive one )"
```

atmosphere.getO2LinesPathLength.html

atmosphere.getO2LinesPathLength - Function

4.1.1 get the integrated O2 lines Path

Description

Retrieve the integrated Atmospheric Phase Delay (due to O2 Lines) along the atmospheric path corresponding to the 1st guess water column for channel nc in spectral window spwid.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
Quantity

Example

atmosphere.getO3LinesPathLength.html

atmosphere.getO3LinesPathLength - Function

4.1.1 get the integrated O3 lines Path

Description

Retrieve the integrated Atmospheric Phase Delay (due to O3 Lines) along the atmospheric path corresponding to the 1st guess water column for channel nc in spectral window spwid.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
Quantity

Example

atmosphere.getCOLinesPathLength.html

atmosphere.getCOLinesPathLength - Function

4.1.1 get the integrated CO lines Path

Description

Retrieve the integrated Atmospheric Path length (due to CO Lines) along the atmospheric path corresponding to the 1st guess water column for channel nc in spectral window spwid.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
Quantity

Example

atmosphere.getN2OLinesPathLength.html

atmosphere.getN2OLinesPathLength - Function

4.1.1 get the integrated N2O lines Path

Description

Retrieve the integrated Atmospheric Path length (due to N2O Lines) along the atmospheric path corresponding to the 1st guess water column for channel nc in spectral window spwid.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns
Quantity

Example

atmosphere.getNonDispersivePathLength.html

atmosphere.getNonDispersivePathLength - Function

4.1.1 get the integrated Atmospheric Non-Dispersive Path

Description

Get the integrated zenith H2O Atmospheric Path length (Non-Dispersive part) for the current conditions, for channel nc in spectral window spwid.

Arguments

Inputs		
nc	Channel number (0-based; defaults to reference channel)	
	allowed:	int
	Default:	-1
spwid	Int standing for spectral window id (0-based)	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
w = at.getUserWH2O()
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
nfR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total Dispersive Delay at ", fC['value'][0], fC['unit'], " for 1.0 air mass: ",\
      at.getDispersivePathLength()['value'][0] /  w['value'][0], " meters per mm of water va
print "(",100*(at.getDispersivePathLength()['value'][0] /  w['value'][0])/(at.getNonDispersi
      "% of the Non-dispersive one )"
```

atmosphere.getAbsH2OLines.html

atmosphere.getAbsH2OLines - Function

4.1.1 Get H2O lines Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

Description

Accessor to get H2O lines Absorption Coefficient at layer nl, spectral window spwid and channel nf.

Arguments

Inputs		
nl	atmospheric layer number	
	allowed:	int
	Default:	
nf	frequency channnel number	
	allowed:	int
	Default:	0
spwid	spectral window id	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
ac = at.getAbsH2OLines(0, 0, 0)
print "H2O lines absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac['uni
```

atmosphere.getAbsH2OCont.html

atmosphere.getAbsH2OCont - Function

4.1.1 Get H2O continuum Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

Description

Get H2O continuum Absorption Coefficient at layer nl, spectral window spwid and frequency channel nf

Arguments

Inputs		
nl	atmospheric layer number	
	allowed:	int
	Default:	
nf	frequency channnel number	
	allowed:	int
	Default:	0
spwid	spectral window id	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
ac = at.getAbsH2OCont(0, 0, 0)
print "H2OCont absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac['unit
```

atmosphere.getAbsO2Lines.html

atmosphere.getAbsO2Lines - Function

4.1.1 Get O2 lines Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

Description

Get O2 lines Absorption Coefficient at layer nl, spectral window spwid and frequency channel nf

Arguments

Inputs		
nl	atmospheric layer number	
	allowed:	int
	Default:	
nf	frequency channnel number	
	allowed:	int
	Default:	0
spwid	spectral window id	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
ac = at.getAbsO2Lines(0, 0, 0)
print "O2 lines absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac['unit
```

atmosphere.getAbsDryCont.html

atmosphere.getAbsDryCont - Function

4.1.1 Get Dry Continuum Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

Description

Get Dry Continuum Absorption Coefficient at layer nl, spectral window spwid and frequency channel nf

Arguments

Inputs		
nl	atmospheric layer number	
	allowed:	int
	Default:	
nf	frequency channnel number	
	allowed:	int
	Default:	0
spwid	spectral window id	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
ac = at.getAbsDryCont(0, 0, 0)
print "Dry Continuum absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac
```

atmosphere.getAbsO3Lines.html

atmosphere.getAbsO3Lines - Function

4.1.1 Get O3 lines Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

Description

Get O3 lines Absorption Coefficient at layer nl, spectral window spwid and frequency channel nf

Arguments

Inputs		
nl	atmospheric layer number	
	allowed:	int
	Default:	
nf	frequency channnel number	
	allowed:	int
	Default:	0
spwid	spectral window id	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
ac = at.getAbsO3Lines(0, 0, 0)
print "O3 lines absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac['unit
```

atmosphere.getAbsCOLines.html

atmosphere.getAbsCOLines - Function

4.1.1 Get CO lines Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

Description

Get CO lines Absorption Coefficient at layer nl, spectral window spwid and frequency channel nf

Arguments

Inputs		
nl	atmospheric layer number	
	allowed:	int
	Default:	
nf	frequency channnel number	
	allowed:	int
	Default:	0
spwid	spectral window id	
	allowed:	int
	Default:	0

Returns
Quantity

Example

```
ac = at.getAbsCOLines(0, 0, 0)
print "CO lines absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac['unit
```

atmosphere.getAbsN2OLines.html

atmosphere.getAbsN2OLines - Function

4.1.1 Get N2O lines Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

Description

Get N2O lines Absorption Coefficient at layer nl, spectral window spwid and frequency channel nf

Arguments

Inputs		
nl	atmospheric layer number	
	allowed:	int
	Default:	
nf	frequency channnel number	
	allowed:	int
	Default:	0
spwid	spectral window id	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
ac = at.getAbsN2OLines(0, 0, 0)
print "N2O lines absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac['uni
```

atmosphere.getAbsTotalDry.html

atmosphere.getAbsTotalDry - Function

4.1.1 Get Total Dry Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

Description

Get total dry Absorption Coefficient at layer nl, spectral window spwid and frequency channel nf

Arguments

Inputs		
nl	atmospheric layer number	
	allowed:	int
	Default:	
nf	frequency channnel number	
	allowed:	int
	Default:	0
spwid	spectral window id	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
ac = at.getAbsTotalDry(0, 0, 0)
print "Total dry absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac['uni
```

atmosphere.getAbsTotalWet.html

atmosphere.getAbsTotalWet - Function

4.1.1 Get total wet absorption coefficient at layer nl and frequency channel nf in spectral window spwid

Description

Get total wet absorption coefficient at layer nl, spectral window spwid and frequency channel nf

Arguments

Inputs		
nl	atmospheric layer number	
	allowed:	int
	Default:	
nf	frequency channnel number	
	allowed:	int
	Default:	0
spwid	spectral window id	
	allowed:	int
	Default:	0

Returns

Quantity

Example

```
ac = at.getAbsTotalWet(0, 0, 0)
print "Total wet absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac['uni
```

atmosphere.setUserWH2O.html

atmosphere.setUserWH2O - Function

4.1.1 set the user zenith water vapor column

Description

Set user zenith water vapor column for forward radiative transfer calculations.

Arguments

Inputs		
wh2o	mm	User water vapor column
		allowed: doublemm
		Default: 0.0

Returns

bool

Example

```
wh2o=qa.quantity(0.8,"mm")
at.setUserWH2O(wh2o)
```

---

atmosphere.getUserWH2O.html

atmosphere.getUserWH2O - Function

4.1.1 get the user zenith water vapor column

Description

Get user zenith water vapor column for forward radiative transfer calculations.

Arguments

Returns
Quantity

Example

```
print "water vapor column: ", at.getUserWH2O()['value'][0], at.getUserWH2O()['unit']
```

---

atmosphere.setAirMass.html

atmosphere.setAirMass - Function

4.1.1 Set the air mass

Description

Setter for air mass in SkyStatus without performing water vapor retrieval.

Arguments

Inputs		
airmass	Air Mass	
	allowed:	double
	Default:	

Returns

bool

Example

```
at.setAirMass(1.51)
```

---

atmosphere.getAirMass.html

atmosphere.getAirMass - Function

4.1.1 Get the air mass

Description

Accessor to get airmass.

Arguments

Returns
double

Example

```
at.setAirMass(2.0)
print "(INPUT CHANGE) Air mass: ", at.getAirMass()
```

---

atmosphere.setSkyBackgroundTemperature.html

atmosphere.setSkyBackgroundTemperature - Function

4.1.1 Set the sky background temperature

Description

Set sky background temperature in SkyStatus without performing water vapor retrieval

Arguments

Inputs			
tbgr	K	sky background temperature	
		allowed:	doubleK
		Default:	2.73

Returns

bool

Example

```
at.setSkyBackgroundTemperature(qa.quantity(2.73,'K'))
```

atmosphere.getSkyBackgroundTemperature.html

atmosphere.getSkyBackgroundTemperature - Function

4.1.1 Get the sky background temperature

Description

Get the sky background temperature

Arguments

Returns
Quantity

Example

```
t = at.getSkyBackgroundTemperature()
print t['value'][0], t['unit']
# 2.73 K
```

---

atmosphere.getAverageTebbSky.html

atmosphere.getAverageTebbSky - Function

4.1.1 Returns average equiv. BB Temp

Description

Returns the average Equivalent Blackbody Temperature in spectral window spwid, for the current conditions and a perfect sky coupling.

Arguments

Inputs		
spwid		Spectral window (0-based) allowed: int Default: 0
wh2o	mm	User specified water column length in mm. Default is not to use wh2o. allowed: doublemm Default: -1

Returns

Quantity

Example

```
wh2o = qa.quantity(0.4,'mm')
print "(INPUT CHANGE) water vapor column:", wh2o['value'], wh2o['unit']
print "(NEW OUTPUT) T_EBB =", at.getAverageTebbSky(0,wh2o)['value'][0], at.getAverageTebbSky
```

atmosphere.getTebbSky.html

atmosphere.getTebbSky - Function

4.1.1 Returns equiv. BB Temp

Description

Gets the Equivalent Blackbody Temperature in spectral window spwid and channel nc, for Water Vapor Column wh2o, the current Air Mass, and perfect Sky Coupling to the sky.

Arguments

Inputs		
nc		Channel number (0-based) - defaults to reference channel
		allowed: int
		Default: -1
spwid		Spectral window (0-based)
		allowed: int
		Default: 0
wh2o	mm	User specified water column length in mm. Default is not to use wh2o.
		allowed: doublemm
		Default: -1

Returns

Quantity

Example

```
for s in range(at.getNumSpectralWindows()):
        for i in range(at.getNumChan(s)):
                print "Band", s, " channel ", i, "TebbSky = ", at.getTebbSky(i,s)['value'][0
```

atmosphere.getTebbSkySpec.html

atmosphere.getTebbSkySpec - Function

4.1.1 Returns equiv. BB Temp on each channel of a band

Description

Gets the Equivalent Blackbody Temperatures in a spectral window spwid for Water Vapor Column wh2o, the current Air Mass, and perfect Sky Coupling to the sky.

Arguments

Outputs		
tebbSky	K	the Equivalent Blackbody Temperatures in a band - Quantum with a vector value and unit of K allowed: doubleK Default:
Inputs		
spwid		Spectral window (0-based) allowed: int Default: 0
wh2o	mm	User specified water column length in mm. Default is not to use wh2o. allowed: doublemm Default: -1

Returns
int

Example

```
sw=at.getWetOpacitySpec()
# returns a tuple of
# 0 - The number of channels, and
# 1 - the Equivalent Blackbody Temperatures in a band
sw[1]['value']
```

```
# [34.687910103670511,
#  35.496193465331679,
#  36.460355664151791,
#  37.419146813713745,
#  37.9452005127634,
#  38.722631196093729,
#  39.593561594172662,
#  40.528694048924017]

sw[0]
# 8

Another example:
for s in range(at.getNumSpectralWindows()):
        print "band", s
        tebbspec = at.getTebbSkySpec(spwid=s)
        for i in range(at.getNumChan(s)):
                print " - TebbSky %f [%s] " % (tebbspec[1]['value'][i],tebbspec[1]['unit'])
```

---

atmosphere.getAverageTrjSky.html

atmosphere.getAverageTrjSky - Function

4.1.1 Returns the average Rayleigh-Jeans Temperature

Description

Returns the average Rayleigh-Jeans Temperature in spectral window spwid, for the current conditions and a perfect sky coupling.

Arguments

Inputs			
spwid		Spectral window (0-based)	
		allowed:	int
		Default:	0
wh2o	mm	User specified water column length in mm. Default is not to use wh2o.	
		allowed:	doublemm
		Default:	-1

Returns

Quantity

Example

```
wh2o = qa.quantity(0.4,'mm')
print "(INPUT CHANGE) water vapor column:", wh2o['value'], wh2o['unit']
print "(NEW OUTPUT) T_RJ =", at.getAverageTrjSky(0,wh2o)['value'][0], at.getAverageTrjSky(0,
```

atmosphere.getTrjSky.html

atmosphere.getTrjSky - Function

4.1.1 Returns the Rayleigh-Jeans Temperature

Description

Gets the Rayleigh-Jeans Temperature in spectral window spwid and channel nc, for Water Vapor Column wh2o, the current Air Mass, and perfect Sky Coupling to the sky.

Arguments

Inputs		
nc		Channel number (0-based) - defaults to reference channel
		allowed: int
		Default: -1
spwid		Spectral window (0-based)
		allowed: int
		Default: 0
wh2o	mm	User specified water column length in mm. Default is not to use wh2o.
		allowed: doublemm
		Default: -1

Returns
Quantity

Example

```
for s in range(at.getNumSpectralWindows()):
        for i in range(at.getNumChan(s)):
                print "Band", s, " channel ", i, "TrjSky = ", at.getTrjSky(i,s)['value'][0]
```

atmosphere.getTrjSkySpec.html

atmosphere.getTrjSkySpec - Function

4.1.1 Returns the Rayleigh-Jeans Temperatures on each channel of a band

Description

Gets the Rayleigh-Jeans Temperatures in a spectral window spwid for Water Vapor Column wh2o, the current Air Mass, and perfect Sky Coupling to the sky.

Arguments

Outputs		
trjSky	K	the Rayleigh-Jeans Temperatures in a band - Quantum with a vector value and unit of K allowed: doubleK Default:
Inputs		
spwid		Spectral window (0-based) allowed: int Default: 0
wh2o	mm	User specified water column length in mm. Default is not to use wh2o. allowed: doublemm Default: -1

Returns
int

Example

```
sw=at.getWetOpacitySpec()
# returns a tuple of
# 0 - The number of channels, and
# 1 - the Equivalent Blackbody Temperatures in a band
sw[1]['value']
```

```
# [34.687910103670511,
#  35.496193465331679,
#  36.460355664151791,
#  37.419146813713745,
#  37.9452005127634,
#  38.722631196093729,
#  39.593561594172662,
#  40.528694048924017]

sw[0]
# 8

Another example:
for s in range(at.getNumSpectralWindows()):
        print "band", s
        trjspec = at.getTrjSkySpec(spwid=s)
        for i in range(at.getNumChan(s)):
                print " - TrjSky %f [%s] " % (trjspec[1]['value'][i],trjspec[1]['unit'])
```

SingleDish.html

Chapter 5

Package SingleDish

Single-dish data analysis package
    sd   sd-Module.html

5.1 sd - Module

Single-dish data analysis package

Description The sd module provides functions and tools to reduce and analyse single-dish data. The module can be categorised as

- sd - Top level tool that contains the following sub tools and general functions for single-dish data reduction and analysis such as calibration, data averaging, etc.
- sd.scantable - Representation of single-dish data format. The tool also contains getter/setter functions as well as functions that manipulates data contents such as baseline fitting, flagging, etc.
- sd.selector - Data selector that operates on scantable.
- sd.fitter - This tool is an engine to perform both baseline and spectral line fitting.
- sd.linefinder - This tool performs automatic line finding according to the user specified thresholds.
- sd.linecatalog - Line catalog that contains name of species, line frequency with error, and line intensity. It accepts both pre-defined line catalog (JPL) for this tool and an user provided catalog in ASCII or specific table format.
- sd.plotter - Plotter tool dedicated for single-dish data

- sd.coordinate - Tool for conversion from pixel to velocity or frequency.
- sd.opacity_model - MIRIAD like opacity model.
- sd.asapgrid - Tool to convolve map data onto regularly spaced grid.
- sd.asaplog - Wrapper object to allow for both casapy and asap logging.

sd-Tool.html

5.1.1 sd - Tool

Single-dish data analysis package
Requires: Synopsis

Description

The sd tool provides data reduction functions for single-dish (auto-correlation) data. It is actually a standalone software that is called ASAP (ATNF Spectral Analysis Package). For more information about ASAP, visit here.
The sd tool is a top level object and it contains various sub tools as well as a number of functions.

Important note
In the latest release, single dish package is loaded at the start-up of CASA. Therefore, it is not necessary to load ASAP explicitly anymore.

Data format
There are various types of data format for single-dish data. Currently, sd tool supports the following data formats for reading and writing.

- reading: Scantable, Measurement Set, SDFITS (GBT, ATNF), NRO
- writing: Scantable, Measurement Set, SDFITS (ATNF), image FITS, CLASS, ASCII

An internal data format for sd tool is Scantable. The Scantable is defined as CASA table and is specially designed for single-dish data. The data is converted to Scantable during the processing whatever its original data format is. The result of the processing can be stored in other data format than Scantable. The Scantable is implemented as sd.scantable object in sd tool. See sd.scantable for details.

Selecting data
The Scantable can contain data with various types of frequency setting, multiple polarization components, and so on. In case you want to select data by some conditions, sd.selector is available for that. The sd.selector provides various functions to select and/or to sort data as well as it provides an interface to select data by TaQL. See sd.selector for details.

Fitting
The sd.fitter is an object to perform fitting data. Both baseline and line fitting for spectral data can be done by this object if appropriate fitting function and masked region are set. Currently, Gaussian and Lorentzian are supported for line fitting, while polynomials with arbitrary order are available for baseline

fitting. Masking should be applied in the form of Bool array. See sd.fitter for details.

Line finding
The sd tool provides automatic line finding functionality that is called sd.linefinder. The sd.linefinder has a few control parameter for line finding. The user is able to customize a behavior of line finding process by changing these parameters. Simplified line finding tool sd.simplelinefinder is also available. See sd.linefinder and sd.simplelinefinder for details.

Line catalog
The sd.linecatalog object is an interface for line catalog. The input data should be provided from the user. Supported format is ASCII table or specific format for sd.linecatalog. CASA package contains a default line catalog that is a part of JPL line catalog and is specific table format for the sd tool (ASAP). Benefit of this object is that it provides an interface to the catalog such as selecting lines by frequency and/or line intensity. It also enables an interaction between spectral data on the sd.plotter such as overlaying line catalog on the plotted spectra. See sd.linecatalog for details.

Plotter
The sd.plotter is a plotter object that is exclusively designed for sd tool. That provides plotting functionalities for spectral data, time variation of azimuth and elevation, and pointing information. For plotting spectral data, it supports multi-panel and multi-page plot. See sd.plotter for details.

Coordinate system
The sd.coordinate is a representation of spectral coordinate of the data (frequency axis). It provides functions for conversion between pixel (channel), frequency, and velocity. See sd.coordinate for details.

Opacity model
In the sd tool, MIRIAD like atmospheric opacity model is implemented as sd.opacity_model. It calculates opacities from given atmospheric conditions (temperature, pressure, and humidity) and elevation. See sd.opacity_model for details.

Gridding
The tool to convolve spectral data onto regularly spaced grid, sd.asapgrid is available. Currently, three convolution kernels, box, prolate-spheroidal, and gaussian, can be used for convolution. See sd.asapgrid for details.

Logging
Although logging system is integrated into CASA logger, sd tool has own logging functions so that you can use it instead of CASA logging functions. Log messages will be displayed in the CASA logger either you use logging functions for CASA or sd tool. See sd.asaplog for details.
Methods

almacal	Calibration function specific for ALMA data

apexcal	Calibration function specific for APEX data
average_time	Averaging data in time
calfs	Calibration function for frequency switched data
calibrate	High level function for calibration
calnod	Calibration function for nodding data
calps	Calibration function for position switched data
commands	Show a list of commands and their short descriptions
dosigref	Equivalent function with dosigref in GBTIDL
dototalpower	Equivalent function with dototalpower in GBTIDL
fitter	Create a fitter object
get_revision	Get revision of the source code for the tool
is_asap_cli	Return True if the tool is launched as standalone software
is_casapy	Return True if the tool is loaded as part of CASA
linecatalog	Create a linecatalog object
linefinder	Create a linefinder object
list_files	Return a list of files readable by sd tool
list_rcparameters	Print a list of rc parameters
list_scans	Return a list of scantables created by the user
mask_and	Logical operation function on array
mask_or	Logical operation function on array
mask_not	Logical operation function on array
merge	Merge a list of scantables into one
opacity_model	Create a opacity_model object
quotient	Take a quotient of a signal and reference scan
rc	Set the current rc parameters
scantable	Create scantable object
selector	Create selector object
skydip	Determine the opacity from a set of 'skydip' observations
splitant	Split Measurement Set data by antenna and save them as scantable
unique	Return the unique values in a list
welcome	Return a welcome message

sd.almacal.html

sd.almacal - Function

5.1.1 Calibration function specific for ALMA data

Description

This method is properly defined for calibration of ALMA data. Input data must be a scantable object. The calibration scheme is,

$$T_{\rm a}^{*} = T_{\rm sys}\frac{ON - OFF}{OFF},$$

where $T_{\rm a}^{*}$ is an antenna temperature, $T_{\rm sys}$ is a system temperature, ON and OFF are raw (uncalibrated) spectral data that correspond to on-source and off-source position, respectively. The OFF scan is linearly interpolated in time if it exists in the vicinity of target ON scan. The calmode argument specifies calibration mode. Supported calibration modes are 'ps' (position switch) and 'fs' (frequency switch). The 'ps' includes calibration for nutator switching as well as classical position switching. The 'ps' mode also supports calibration of OTF position raster scanned data that consists of on-source scans with simple scan pattern and explicit off-source scans.

Arguments

Inputs		
scantab	input data as a scantable	
	allowed:	scantable
	Default:	
scannos	a list of scan numbers to be calibrated	
	allowed:	(list of) integer, string
	Default:	[] (all data)
calmode	Calibration mode	
	allowed:	string ('ps', 'fs', 'none')
	Default:	'none'
verify	Verify calibration if True	
	allowed:	bool
	Default:	False

Returns
scantable

Example

```
s=sd.scantable('alma-scans.asap',average=False,getpt=True)
scal=sd.almacal(s,calmode='ps')
```

sd.apexcal.html

sd.apexcal - Function

5.1.1 Calibration function specific for APEX data

Description

This method is properly defined for calibration of APEX data. Input data must be a scantable object. The calibration scheme is essentially same as ALMA data,

$$T_{\rm a}^{*} = T_{\rm sys} \frac{ON - OFF}{OFF},$$

where $T_{\rm a}^{*}$ is an antenna temperature, $T_{\rm sys}$ is a system temperature, ON and OFF are raw (uncalibrated) spectral data that correspond to on-source and off-source position, respectively. Only difference with ALMA calibration is that $T_{\rm sys}$ doesn't provided. Instead, it is computed from a calibration temperature and two calibration scans: a blank sky and a load with known temperature. The OFF scan is linearly interpolated in time if it exists in the vicinity of target ON scan. The calmode argument specifies calibration mode. Supported calibration modes are 'ps' (position switch) and 'fs' (frequency switch). The 'ps' includes calibration for nutator switching as well as classical position switching.

Arguments

Inputs		
scantab	input data as a scantable	
	allowed:	scantable
	Default:	
scannos	a list of scan numbers to be calibrated	
	allowed:	(list of) integer, string
	Default:	[] (all data)
calmode	Calibration mode	
	allowed:	string ('ps', 'fs', 'none')
	Default:	'none'
verify	Verify calibration if True	
	allowed:	bool
	Default:	False

Returns
scantable

Example

```
s=sd.scantable('alma-scans.asap',average=False,getpt=True)
scal=sd.apexcal(s,calmode='ps')
```

sd.average_time.html

sd.average_time - Function

5.1.1 Averaging data in Time

Description

The function computes a weighted time average of a scantable or a list of scantables. The averaging is done in channel only. Supported weighting schemes are as follows:

- none — no weighting
- var — 1/var(spectrum) weighted
- tsys — 1/Tsys**2 weighted
- tint — integration time weighted
- tintsys — Tint/Tsys**2 weighted
- median — median averaging

Arguments

Inputs		
	Input data as a scantable	
	allowed:	scantable or a list of scantables
	Default:	
mask	An optional mask (only used for 'var' and 'tsys' weighting)	
	allowed:	bool list
	Default:	none
scanav	True averages each scan separately, False averages all scans together	
	allowed:	bool
	Default:	False
weight	Weighting scheme (see description)	
	allowed:	string
	Default:	'tint'
align	Align the spectra in velocity before averaging. It takes the time of the first spectrum in the first scantable as reference time.	
	allowed:	bool
	Default:	False

Returns
scantable

Example

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

sd.calfs.html

sd.calfs - Function

5.1.1 Calibration function for frequency switched data

Description

Calibrate GBT frequency switched data. Adopted from GBTIDL getfs. Currently calfs identify the scans as frequency switched data if source type enum is fson and fsoff. The data must contains 'CAL' signal on/off in each integration. To identify 'CAL' on state, the source type enum of foncal and foffcal need to be present.

Arguments

Inputs		
scantab	Input data as a scantable	
	allowed:	scantable
	Default:	
scannos	A list of scan numbers to be calibrated	
	allowed:	(list of) integer
	Default:	[] (all data)
smooth	Optional box smoothing order for the reference	
	allowed:	int
	Default:	1 (no smoothing)
tsysval	Optional user specified Tsys	
	allowed:	float
	Default:	0.0 (use Tsys in the data)
tauval	Optional user specified optical depth	
	allowed:	float
	Default:	0.0
verify	Verify calibration if True	
	allowed:	bool
	Default:	False

Returns
scantable

Example

```
s=sd.scantable('FLS3a_HI.asap')                # load in the saved ASAP dataset with FLS
scanns = s.getscannos()                        # get a list of the scan numbers in the s
sn=list(scanns)
res=sd.calfs(s,sn)                             # Do a frequency switched calibration on
```

sd.calibrate.html

sd.calibrate - Function

5.1.1 High level function for calibration that calls appropriate calibration function depending on the data

Description

This method is a high level function for calibration. It calls appropriate calibration function depending on the origin of the data. It checks an calmode argument first, and then looks antenna name in the data. Calibration functions that can be called from the method are:

- calnod if calmode is 'nod'
- auto_quotient if calmode is 'quotient'
- calps if calmode is 'ps' and antenna name is 'GBT'
- calfs if calmode is 'fs' and antenna name is 'GBT'
- apexcal if calmode is 'ps' or 'fs' and antenna name contains 'APEX'
- almacal if calmode is 'ps', 'fs', or 'otf' and antenna name contains 'ALMA' or 'OSF'

Arguments

Inputs		
scantab	input data as a scantable	
	allowed:	scantable
	Default:	
scannos	a list of scan numbers to be calibrated	
	allowed:	(list of) integer, string
	Default:	[] (all data)
calmode	Calibration mode	
	allowed:	string ('ps', 'fs', 'none')
	Default:	'none'
verify	Verify calibration if True	
	allowed:	bool
	Default:	False

Returns
scantable

Example

```
s=sd.scantable('alma-scans.asap',average=False,getpt=True)
scal=sd.calibrate(s,calmode='ps')
```

sd.calnod.html

sd.calnod - Function

5.1.1 Calibration function for nodding data

Description

This method performs full (but a pair of scans at time) processing of GBT Nod data calibration. Adopted from GBTIDL's getnod

Arguments

Inputs		
scantab	Input data as a scantable	
	allowed:	scantable
	Default:	
scannos	A pair of scan numbers, or the first scan number of the pair	
	allowed:	(list of) integer
	Default:	[] (all data)
smooth	Box car smoothing order	
	allowed:	int
	Default:	1 (no smoothing)
tsysval	Optional user specified Tsys	
	allowed:	float
	Default:	0.0 (use Tsys in the data)
tauval	Optional user specified optical depth (not implemented yet)	
	allowed:	float
	Default:	0.0
tcalval	Optional user specified Tcal	
	allowed:	float
	Default:	0.0
verify	Verify calibration if True	
	allowed:	bool
	Default:	False

Returns
scantable

Example

```
s=sd.scantable('IRC+10216_rawACSmod',False)#load the data without averaging
scal=sd.calnod(s,[229,230])             # Calibrate CS scans
```

sd.calps.html

sd.calps - Function

5.1.1 Calibration function for position switched data

Description

The method calibrates GBT position switched data Adopted from GBTIDL getps Currently calps identify the scans as position switched data if source type enum is pson or psoff. The data must contains 'CAL' signal on/off in each integration. To identify 'CAL' on state, the source type enum of poncal and poffcal need to be present.

Arguments

Inputs		
scantab	Input data as a scantable	
	allowed:	scantable
	Default:	
scannos	A list of scan numbers to be calibrated	
	allowed:	(list of) integer
	Default:	[] (all data)
smooth	Optional box smoothing order for the reference	
	allowed:	int
	Default:	1 (no smoothing)
tsysval	Optional user specified Tsys	
	allowed:	float
	Default:	0.0 (use Tsys in the data)
tauval	Optional user specified optical depth	
	allowed:	float
	Default:	0.0
tcalval	Optional user specified Tcal	
	allowed:	float
	Default:	0.0 (use Tcal in the data)
verify	Verify calibration if True	
	allowed:	bool
	Default:	False

Returns
scantable

Example

```
s=sd.scantable('OrionS_rawACSmod',False)#load the data without averaging
scal=sd.calps(s,[24,25,26,27])          # Calibrate SiO scans
```

sd.commands.html

sd.commands - Function

5.1.1 Show a list of commands and their short descriptions

Description

The method prints a list of commands and their short descriptions. The output is sorted by their intents and/or subtools that the commands associate.

Arguments

Returns

Example

```
sd.commands()
# Output will be as follows
[The scan container]
    scantable           - a container for integrations/scans
                          (can open asap/rpfits/sdfits and ms files)
        copy            - returns a copy of a scan
        get_scan        - gets a specific scan out of a scantable
                          (by name or number)
        drop_scan       - drops a specific scan out of a scantable
                          (by number)
        set_selection   - set a new subselection of the data
        get_selection   - get the current selection object
        summary         - print info about the scantable contents
        stats           - get specified statistic of the spectra in
                          the scantable
        stddev          - get the standard deviation of the spectra
                          in the scantable
        get_tsys        - get the TSys
        get_time        - get the timestamps of the integrations
        get_inttime     - get the integration time
        get_sourcename  - get the source names of the scans
        get_azimuth     - get the azimuth of the scans
```

```
get_elevation   - get the elevation of the scans
get_parangle    - get the parallactic angle of the scans
get_coordinate  - get the spectral coordinate for the given row,
                  which can be used for coordinate conversions
get_weather     - get the weather condition parameters
get_unit        - get the current unit
set_unit        - set the abcissa unit to be used from this
                  point on
...
```

sd.dosigref.html

sd.dosigref - Function

5.1.1 Equivalent function with dosigref in GBTIDL

Description

The method calculates a quotient (sig-ref/ref * Tsys). Adopted from GBTIDL dosigref.

Arguments

Inputs		
sig	On-source data as a scantable	
	allowed:	scantable
	Default:	
ref	Reference data as a scantable	
	allowed:	scantable
	Default:	
smooth	Width of box car smoothing for reference	
	allowed:	int
	Default:	1 (no smoothing)
tsysval	User specified Tsys	
	allowed:	float
	Default:	0.0 (use Tsys in the data)
tauval	User specified optical depth (required if tsysval is set)	
	allowed:	float
	Default:	0.0

Returns
scantable

Example

```
s = sd.scantable('OrionS_rawACSmod',average=False)
sel = sd.selector()
# calibration scans
sel.set_types([srctype.poncal,srctype.poffcal])
s.set_selection(sel)
ssubon=s.copy()
s.set_selection()
```

```
sel.reset()
# off-calibration scans
sel.set_types([srctype.pson,srctype.psoff])
s.set_selection(sel)
ssuboff=s.copy()
s.set_selection()
sel.reset()
# calibration
cals = sd.dototalpower(ssubon,ssuboff)
# ON scan
sel.set_types(srctype.pson)
cals.set_selection(sel)
sig = cals.copy()
cals.set_selection()
sel.reset()
# OFF scan
sel.set_types(srctype.psoff)
cals.set_selection(sel)
ref = cals.copy()
cals.set_selection()
sel.reset()
# get calibrated data
ress = sd.dosigref(sig,ref,smooth=1)
```

sd.dototalpower.html

sd.dototalpower - Function

5.1.1 Equivalent function with dototalpower in GBTIDL

Description

The method performs calibration for CAL on,off signals. Adopted from GBTIDL dototalpower.

Arguments

Inputs		
calon	The 'cal on' subintegration as a scantable	
	allowed:	scantable
	Default:	
caloff	The 'cal off' subintegration as a scantable	
	allowed:	scantable
	Default:	
tcalval	User supplied Tsys	
	allowed:	float
	Default:	0.0 (use Tcal in the data)

Returns
scantable

Example

```
s = sd.scantable('OrionS_rawACSmod',average=False)
sel = sd.selector()
# calibration scans
sel.set_types([srctype.poncal,srctype.poffcal])
s.set_selection(sel)
ssubon=s.copy()
s.set_selection()
sel.reset()
# off-calibration scans
sel.set_types([srctype.pson,srctype.psoff])
s.set_selection(sel)
ssuboff=s.copy()
s.set_selection()
```

```
sel.reset()
# calibration
cals = sd.dototalpower(ssubon,ssuboff)
# ON scan
sel.set_types(srctype.pson)
cals.set_selection(sel)
sig = cals.copy()
cals.set_selection()
sel.reset()
# OFF scan
sel.set_types(srctype.psoff)
cals.set_selection(sel)
ref = cals.copy()
cals.set_selection()
sel.reset()
# get calibrated data
ress = sd.dosigref(sig,ref,smooth=1)
```

sd.get_revision.html

sd.get_revision - Function

5.1.1 Get revision of the source code for the tool

Description

Get the revision of the software. Actually it returns a revision number of the source code that is managed by subversion.

Arguments

Returns
string

Example

```
rev=sd.get_revision()
print rev
'13018'
```

sd.is_asap_cli.html

sd.is_asap_cli - Function

5.1.1 Check if the tool is loaded as part of CASA or is launched as standalone software

Description

The method checks if sd tool (ASAP) is running standalone. This always returns False if you use CASA.

Arguments

Returns

bool (False)

Example

```
isasap=sd.is_asap_cli()
print isasap
False
```

sd.is_casapy.html

sd.is_casapy - Function

5.1.1 Check if the tool is loaded as part of CASA or is launched as standalone software

Description

The method checks if sd tool is running on CASA. This always returns True if you use CASA.

Arguments

Returns
bool (True)

Example

```
iscasa=sd.is_casapy()
print iscasa
True
```

sd.list_files.html

sd.list_files - Function

5.1.1 Return list of files readable by sd tool

Description

Return a list files readable by asap, such as MS, rpf, sdfits, mbf, asap. The method looks a directory that is indicated by path argument, and searches files with extension specified by suffix. Allowed extensions are:

- rpf — RPFITS (default)
- rpf.1 — RPFITS
- rpf.2 — RPFITS
- sdf — SDFITS
- sdfits – SDFITS
- mbf — MBFITS
- asap — Scantable
- ms — Measurement Set

Note that the method just checks the extension of the file name. Thus, the returned list doesn't contains files that don't have the above extensions even if they are readable.

Arguments

Inputs		
path	The directory to list	
	allowed:	string
	Default:	'./' (current directory)
suffix	The file extension	
	allowed:	string
	Default:	'rpf'

Returns

string array

Example

```
files = sd.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']
```

sd.list_rcparameters.html

sd.list_rcparameters - Function

5.1.1 Print a list of rc parameters

Description

Print a list of rc parameters and its default values that determine basic behavior of the tool. The user can be accessed rc parameters directory since the rc parameters are available as a Python dictionary sd.rcParams. There is also a method to set rc parameters called rc. Contents of the rc paramters are described below.

- verbose
  It will disable exceptions and just print the messages (only valid in standalone mode)
- useplotter
  Preload a default plotter
- insitu
  Apply operations on the input scantable or retuern new one
- plotter.gui
  Do we want a GUI or plot to a file
- plotter.stacking
  Default mode for color stacking
- plotter.panelling
  Default mode for panelling
- plotter.ganged
  Push panels together, to share axis labels
- plotter.decimate
  Decimate the number of points plotted by a factor of nchan/1024
- plotter.colours
  Default color
- plotter.linestyles
  Default linestyles
- plotter.histogram
  Enable/disable histogram plotting
- plotter.papertype
  Postscript paper type

- plotter.axesformatting
  The formatting style of the x-axis
- scantable.storage
  Default storage of scantable ('memory'/'disk')
- scantable.history
  Write history of each call to scantable
- scantable.save
  Default output format when saving
- scantable.autoaverage
  Auto averaging on read
- scantable.freqframe
  Default frequency frame to set when function set_freqframe is called
- scantable.verbosesummary
  Control the level of information printed by summary
- scantable.reference
  Control the identification of reference (off) scans (has to be regular expression)
- scantable.parallactify
  Indicate whether the data was parallactified (total phase offset is 0.0)

Arguments

Returns
string

Example

```
sd.list_rcparameters()
# output will be as follows
# general
# only valid in asap standard mode not in scripts or casapy
# It will disable exceptions and just print the messages
verbose                    : True

# preload a default plotter
useplotter                 : True
```

```
# apply operations on the input scantable or return new one
insitu                      : True

# plotting

# do we want a GUI or plot to a file
plotter.gui                 : True

# default mode for colour stacking
plotter.stacking            : Pol

# default mode for panelling
plotter.panelling           : scan

# push panels together, to share axis labels
plotter.ganged              : True

...
```

sd.list_scans.html

sd.list_scans - Function

5.1.1 Return a list of scantables created by the user

Description

The method prints and returns a list of scantables that the user created.

Arguments

Returns

Example

```
# no scantable is created yet
scanlist=sd.list_scans()
print scanlist
[]
# create scantable
s=sd.scantable('OrionS_rawACSmod',average=False)
# run list_scans() again
scanlist=sd.list_scans()
print scanlist
['s']
```

sd.mask_and.html

sd.mask_and - Function

5.1.1 Logical operation function on array

Description

This is an utility function that performs logical 'and' operation on specified two boolean arrays in element-by-element manner. Input arrays should have same length.

Arguments

Inputs		
a	Input boolean array	
	allowed:	bool array
	Default:	
b	Input boolean array	
	allowed:	bool array
	Default:	

Returns

bool array

Example

```
a=[True,False,False]
b=[True,True,False]
sd.mask_and(a,b)
[True,False,False]
```

sd.mask_or.html

sd.mask_or - Function

5.1.1 Logical operation function on array

Description

This is an utility function that performs logical 'or' operation on specified two boolean arrays in element-by-element manner. Input arrays should have same length.

Arguments

Inputs		
a	Input boolean array	
	allowed:	bool array
	Default:	
b	Input boolean array	
	allowed:	bool array
	Default:	

Returns

bool array

Example

```
a=[True,False,False]
b=[True,True,False]
sd.mask_or(a,b)
[True,True,False]
```

sd.mask_not.html

sd.mask_not - Function

5.1.1 Logical operation function on array

Description

This is an utility function that performs logical 'not' operation on the input boolean array in element-by-element manner.

Arguments

Inputs		
a	Input boolean array	
	allowed:	bool array
	Default:	

Returns
bool array

Example

```
a=[True,False,False]
sd.mask_not(a)
[False,True,True]
```

sd.merge.html

sd.merge - Function

5.1.1 Merge a list of scantables into one

Description

Merge a list of scanatables, or comma-sperated scantables into one scnatble.

Arguments

Inputs		
	Input scantables	
	allowed:	list of scantables or comma-separated scantables
	Default:	

Returns

scantable

Example

```
scan1 = sd.scantable('scan1.asap')
scan2 = sd.scantable('scan2.asap')
myscans = [scan1, scan2]
allscans = sd.merge(myscans)
# or equivalent
sameallscans = sd.merge(scan1, scan2)
```

sd.quotient.html

sd.quotient - Function

5.1.1 Take a quotient of a signal and reference scan

Description

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. The preserve argument controls if continuum is preserved or not. The equation used in the method depends on its value. If preserve is True, the equation is,

$$T_{\rm a}^{*} = T_{\rm sys}^{\rm OFF} \frac{ON}{OFF} - T_{\rm sys}^{\rm OFF},$$

while if preserve is False,

$$T_{\rm a}^{*} = T_{\rm sys}^{\rm OFF} \frac{ON}{OFF} - T_{\rm sys}^{\rm ON},$$

where $T_{\rm a}^{*}$ is antenna temperature, ON and OFF are raw (uncalibrated) spectral data that correspond to on-source and off-source position, $T_{\rm sys}^{\rm ON}$ and $T_{\rm sys}^{\rm OFF}$ are system temperatures of ON and OFF scans, respectively.

Arguments

Inputs		
source	The on-source scan as a scantable	
	allowed:	scantable
	Default:	
reference	The reference scan as a scantable	
	allowed:	scantable
	Default:	
preserve	Preserve the continuum or remove it	
	allowed:	bool
	Default:	True

Returns
scantable

Example

```
s = sd.scantable('OrionS_rawACSmod',average=False)
sel = sd.selector()
# ON scan
sel.set_types(srctype.pson)
s.set_selection(sel)
sig = s.copy()
s.set_selection()
sel.reset()
# OFF scan
sel.set_types(srctype.psoff)
s.set_selection(sel)
ref = s.copy()
s.set_selection()
sel.reset()
# get quotient data
cals = sd.quotient(source=sig,reference=ref,preserve=True)
```

sd.rc.html

sd.rc - Function

5.1.1 Set the current rc parameters

Description

Set the current rc parameters (sd.rcParams, see list_rcparameters). The 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

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

sets the current rc params and is equivalent to

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

Arguments

Inputs		
group	Grouping for the rc	
	allowed:	string ('','scantable','plotter')
	Default:	
	A list of attribute name/value pairs	
	allowed:	comma separated list of name/value pairs
	Default:	

Returns

Example

```
# set scantable.save as 'SDFITS' and scantable.storage as 'disk'
sd.rc('scantable',save='SDFITS', storage='disk')
# set insitu as False
sd.rc('',insitu=False)
```

sd.skydip.html

sd.skydip - Function

5.1.1 Determine the opacity from a set of 'skydip' observations

Description

Determine the opacity from a set of 'skydip' obervations. This can be any set of observations over a range of elevations, but will ususally be a dedicated (set of) scan(s). Return a list of 'n' opacities for 'n' IFs. In case of averagepol being 'False' a list of 'n*m' elements where 'm' is the number of polarisations, e.g. nIF = 3, nPol = 2 ⇒ [if0pol0, if0pol1, if1pol0, if1pol1, if2pol0, if2pol1] The opacity is determined by fitting a first order polynomial to:

$$T_{\rm sys}(airmass) = p_0 + airmass * p1,$$

where

$$airmass = \frac{1}{\sin(elevation)}$$

$$\tau = \frac{p_1}{T_{\rm sky}}$$

Arguments

Inputs	
data	File name or scantable or list of them
	allowed: string, string array, scantable, list of scantables
	Default:
averagepol	Return opacity values per polarization or average of opacities for the polarizations
	allowed: bool
	Default: True
tsky	The sky temperature
	allowed: float
	Default: 300.0
plot	Plot each fit (airmass versus Tsys)
	allowed: bool
	Default: False

Returns

float array

Example

```
s=sd.scantable('skydip.asap',average=False)
sd.skydip(data=s,averagepol=True,plot=True)
```

sd.splitant.html

sd.splitant - Function

5.1.1 Split Measurement Set data by antenna and save them as scantable

Description

Split Measurement Set by antenna name, save data as a scantables, and return a list of filename. Notice this method can only be available from CASA. The outprefix argument specifies a prefix of output scantable name. The names of output scantables will be 'outprefix.antenna1.asap', 'outprefix.antenna2.asap', ... where antenna1 and antenna2 is antenna name that are stored in ANTENNA subtable of input Measurement Set.

Arguments

Inputs		
filename	The name of Measurement Set to be read	
	allowed:	string
	Default:	
outprefix	The prefix of output scantable name	
	allowed:	string
	Default:	”
overwrite	If the file should be overwritten if it exists	
	allowed:	bool
	Default:	False

Returns
string array

Example

```
# assume that input MS contains data from 'DV01' and 'PM03'
outfiles=sd.splitant('osfdata.ms',outprefix='test')
print outfiles
['test.DV01.asap','test.PM03.asap']
```

sd.unique.html

sd.unique - Function

5.1.1 Return the unique values in a list

Description

This is an utility function that returns the unique values in a list

Arguments

Inputs		
x	The list to reduce	
	allowed:	array
	Default:	

Returns

any array

Example

```
x=[1,2,3,3,4]
y=sd.unique(x)
print y
[1,2,3,4]
```

sd.welcome.html

sd.welcome - Function

5.1.1 Return a welcome message

Description

Return a welcome message.
Note that the message assumes the sd tool (ASAP) is used as a standalone software so that some informations are not fit with CASA. For example, you should also report bugs on sd tool to CASA helpdesk, and you have to run 'sd.commands() instead of 'commands()' if you want to get a list of commands from CASA console.

Arguments

Returns
string

Example

```
msg=sd.welcome()
print msg
# output will be as follows
Welcome to ASAP vtrunk (2010-08-13) - the ATNF Spectral Analysis Package

Please report any bugs via:
http://svn.atnf.csiro.au/trac/asap/simpleticket

[IMPORTANT: ASAP is 0-based]
Type commands() to get a list of all available ASAP commands.
```

sd.scantable-Tool.html

5.1.2 sd.scantable - Tool

Representation of single-dish data format

Description

The scantable is a representation of the Scantable, which is a data format designed properly for single-dish data. The scantable is implemented as CASA table so that it is possible to access data via table tool. On the other hand, the scantable has its own functions to get/set some data. In addition to the simple setter/getter and utility functions, the scantable has some data reduction function such as calibration, baseline fitting, etc.
The 'insitu' option often appears in the functions. This parameter controls if a certain operation is applied to this scantable (true) or return another scantable that is a result of the operation (false). In former case, the original scantable loaded will be lost although the data on disk will be kept. The default value of the option depends on sd.rcParams dictionary. See sd.list_rcparameters for details about sd.rcParams.
The constructor takes several arguments that handles some of optional behavior of it. The getpt and antenna arguments are only effective if file specified by filename argument is Measurement Set format.

Definition
A documentation for detailed definition of the Scantable will be prepared by ATNF.

Arguments

Inputs		
filename	Name of an input file, or a reference to an existing scantable (advanced)	
	allowed:	string, scantable
	Default:	
average	Average all integrations within a scan on read	
	allowed:	bool
	Default:	None (taken from scantable.autoaverage attribute of rc parameters)
unit	Brightness unit. It must be consistent with K or Jy. It overrides the default value or replaces the value in existing scantables	
	allowed:	string
	Default:	None
getpt	Measurement Set input data only. If True, all pointing data are filled.	
	allowed:	bool
	Default:	False
antenna	Measurement Set input data only. Antenna selection by ID or name	
	allowed:	string, integer
	Default:	” (first antenna in the ANTENNA table)
parallactify	Indicate that the data had been parallactified.	
	allowed:	bool
	Default:	None (taken from scantable.parallactify attribute of rc parameters)

Example

```
# create scantable from the data from the second antenna in Measurement Set
s=sd.scantable('sddata.ms',average=False,getpt=True,antenna=1)
```

Methods

add	Return a scan where all spectra have the offset added
auto_cspline_baseline	Perform automatic line finding and baseline subtraction using cubic spline function
auto_poly_baseline	Perform automatic line finding and baseline subtraction using polynomial function
auto_sinusoid_baseline	Perform automatic line finding and baseline subtraction using sinusoidal function
auto_quotient	Automatic quotient for coordinated scans
average_beam	Average beams together for multi-beam observation
average_pol	Average polarizations together
average_time	Return time average of a scan
bin	Perform binning of spectra

chan2data	Return channel/frequency/velocity and spectral value
clip	Flag data by its spectral value
convert_flux	Return a scan where all spectra are converted to Jy or K
convert_pol	Convert data to a different polarization type
copy	Return a copy of this scantable
create_mask	Return a bool array based on [min,max] windows
cspline_baseline	Perform a baseline subtraction using cubic spline function
drop_scan	Return a new scantable where the specified scan number(s) dropped
flag	Apply FFT to the spectra
flag	Flag selected data using specified mask (channel based flag)
flag_nans	Flag NaN values
flag_row	Flag spectra based on specified rows (row based flag)
freq_align	Perform frequency alignment
freq_switch	Apply frequency switching to the data
gain_el	Apply gain-elevation correction based on user-provided data
get_abcissa	Get the abcissa values
get_antennaname	Get antenna name
get_azimuth	Get a list of azimuth during the observation
get_column_names	Get a list of column names in the main table
get_coordinate	Return the spectral coordinate for a given row as a coordinate object
get_direction	Get a list of positions on the sky as a string
get_directionval	Get a list of positions on the sky as a float
get_elevation	Get a list of elevation during the observation
get_fit	Get the stored fits for a row in scantable
get_fluxunit	Get a flux unit
get_inttime	Get a list of integration times for the observation
get_mask	Get mask for the specified row as a bool list
get_mask_indices	Compute a lists of mask start/end indices from the given bool array
get_masklist	Compute a list of mask windows from the given bool array
get_parangle	Get a list of parallactic angles for the observation
get_restfreqs	Get the rest frequency(s) stored in the scantable
get_rms	Calculate rms of the spectrum
get_row	Return a scantable with single row
get_row_selector	Return a selector object that only selects target row
get_scan	Return a specified scan(s) specified by scan number or source name
get_selection	Get current selection that is currently set on this scantable
get_sourcename	Get a list of source names for the observation
get_spectrum	Get the spectrum for the current row
get_time	Get a list of time stamps for the observation
get_tsys	Get a list of system temperatures
get_unit	Get the default unit of spectral axis
get_weather	Get the weather informations
getbeam	Get beam number of the given row
getbeamnos	Get a list of beam numbers in the scantable
getcycle	Get cycle number of the given row
getif	Get IF number of the given row

getifnos	Get a list of IF nubers in the scantable
getmolnos	Get a list of molecule ids in the scantable
getpol	Get polarization number of the given row
getpolnos	Get a list of polarization numbers in the scantable
getscan	Get scan number of the given row
getscannos	Get a list of scan numbers in the scantable
history	Print a history
invert_phase	Invert the phase of the complex polarization
lag_flag	Perform Fourier filtering on the spectra
mx_quotient	Form a quotient using "off" beams when observing in "MX" mode
nbeam	Return a number of beams
nchan	Return a number of channels
ncycle	Return a number of cycles
nif	Return a number of IFs
npol	Return a number of polarizations
nrow	Return a number of rows
nscan	Return a number of scans
opacity	Apply an opacity correction
parallactify	Set a flag to indicate the data should be treated as "parallactified"
poltype	Get a polarization type
poly_baseline	Perform a baseline subtraction using polynomial function
recalc_azel	Recalculate azimuth and elevation for each sky position
resample	Perform a binning
rotate_linpolphase	Rotate a phase of the complex polarization
rotate_xyphase	Rotate a phase of the XY correlation
save	Store the scantable on disk
scale	Scale spectra by the given factor
set_dirframe	Set the frame type of the direction on the sky
set_doppler	Set definition of the Doppler correction
set_feedtype	Set the feed type
set_fluxunit	Set flux unit
set_freqframe	Set the frame type of the spectral axis
set_instrument	Set antenna name
set_restfreqs	Set rest frequency
set_selection	Select a subset of the data
set_sourcetype	Set the types of source to be source or reference scan
set_spectrum	Set spectrum for specified row
set_unit	Set unit for spectral axis
shift_refpix	Shift the reference pixel of the spectral coordinate
sinusoid_baseline	Perform a baseline subtraction using sinusoidal function
smooth	Smooth the spectra
stats	Compute specified statistics of the spectra
stddev	Compute standard deviation of the spectra
summary	Print a summary of the contents of the scantable
swap_linears	Swap the linear polarizations XX and YY

sd.scantable.add.html

sd.scantable.add - Function

5.1.2 Return a scan where all spectra have the offset added

Description

Return a scan where all spectra have the offset added. If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.

Arguments

Inputs		
offset	The offset	
	allowed:	float
	Default:	
insitu	If False a new scantable is returned	
	allowed:	bool
	Default:	None (use default value)

Returns
scantable

Example

sd.scantable.auto_cspline_baseline.html

sd.scantable.auto_cspline_baseline - Function

5.1.2 Perform automatic line finding and baseline subtraction using cubic spline function

Description

Return a scan which has been baselined (all rows) by cubic spline function (piecewise cubic polynomial). Fit will be done with 'sigma-clipping'.
Spectral lines are detected first using linefinder and masked out to avoid them affecting the baseline solution.
The edge argument is 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).
The threshold and chan_avg_limit arguments are linefinder options. The former is the threshold used by line finder. It is better to keep it large as only strong lines affect the baseline solution. The later is a maximum number of consequtive 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. See linefinder for more details on these options.
Note: The best-fit parameter values output in logger and/or blfile are now based on specunit of 'channel'.

Arguments

Inputs	
insitu	If False a new scantable is returned allowed: bool Default: None (use default value)
mask	An optional mask retreived from scantable allowed: bool Default: None (no mask)
npiece	Number of pieces allowed: integer Default: 2
clipthresh	Clipping threshold in unit of sigma allowed: float Default: 3.0
clipniter	Maximum number of iteration of clipping allowed: integer Default: 0
edge	An optional number of channel to drop at the edge of spectrum allowed: integer, integer array Default: (0,0)
threshold	The threshold used by line finder allowed: float Default: 3
chan_avg_limit	A maximum number of consequtive spectral channels to average during the search allowed: int Default: 1
plot	Plot the fit and the residual (currently unavailable) allowed: bool Default: False
getresidual	If False, return best-fit value instead of residual allowed: bool Default: True
showprogress	Show progress status for large data allowed: bool Default: True
minnrow	Minimum number of spectra to show progress status allowed: integer Default: 1000
outlog	Output the coefficients of the best-fit function to logger allowed: bool Default: False
blfile	Name of text file in which the best-fit parameter values to be written allowed: string Default: ”

Returns
scantable

Example

```
scan = sd.scantable('OrionS_rawACSmod_cal',average=False)
bscan = scan.auto_cspline_baseline(npiece=3, insitu=False)
```

sd.scantable.auto_poly_baseline.html

sd.scantable.auto_poly_baseline - Function

5.1.2 Perform automatic line finding and baseline subtraction using polynomial function

Description

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.
The edge argument is 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).
The threshold and chan_avg_limit arguments are linefinder options. The former is the threshold used by line finder. It is better to keep it large as only strong lines affect the baseline solution. The later is a maximum number of consequtive 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. See linefinder for more details on these options.
You can verify and decide whether you apply the fit result or not, if plot argument is True. In that case, you have to answer 'y' or 'n' for each spectra so that setting True is not recommended for large dataset.
Note: The best-fit parameter values output in logger and/or blfile are now based on specunit of 'channel'.

Arguments

Inputs		
mask	An optional mask retreived from scantable	
	allowed:	bool array
	Default:	None (no mask)
order	The order of the polynomial	
	allowed:	integer
	Default:	0
edge	An optional number of channel to drop at the edge of spectrum	
	allowed:	integer, integer array
	Default:	(0,0)
threshold	The threshold used by line finder	
	allowed:	float
	Default:	3
chan_avg_limit	A maximum number of consequtive spectral channels to average during the search	
	allowed:	int
	Default:	1
plot	Plot the fit and the residual.	
	allowed:	bool
	Default:	False
insitu	If False a new scantable is returned	
	allowed:	bool
	Default:	None (use default value)
getresidual	If False, return best-fit value instead of residual	
	allowed:	bool
	Default:	True
showprogress	Show progress status for large data	
	allowed:	bool
	Default:	True
minnrow	Minimum number of spectra to show progress status	
	allowed:	integer
	Default:	1000
outlog	Output the coefficients of the best-fit function to logger	
	allowed:	bool
	Default:	False
blfile	Name of text file in which the best-fit parameter values to be written	
	allowed:	string
	Default:	”

Returns
scantable

Example

```
scan = sd.scantable('OrionS_rawACSmod_cal',average=False)
scan2 = scan.auto_poly_baseline(order=7, insitu=False)
```

sd.scantable.auto_quotient.html

sd.scantable.auto_quotient - Function

5.1.2 Automatic quotient for coordinated scans

Description

This function allows to build quotients automatically. It assumes the observation to have the same number of "ons" and "offs".
The formula to get result depends on the preserve paramter. If it is True, the continuum will be preserved while if it is False, the continuum will be removed. The equation used are

$$T_{\rm a}^{*} = T_{\rm sys}^{\rm OFF} \frac{ON}{OFF} - T_{\rm sys}^{\rm OFF},$$

if preserve is True, while

$$T_{\rm a}^{*} = T_{\rm sys}^{\rm OFF} \frac{ON}{OFF} - T_{\rm sys}^{\rm ON},$$

if preserve is False.
The mode argument controls the on/off decition mode. If mode is 'paired' (default), it identifies 'off' scans by the trailing '_R' (Mopra/Parkes) or '_e'/'_w' (Tid) and matches on/off pairs from the observing pattern. On the other hand, 'time' finds the closest off in time.
Note that the verify argument is not yet implemented.

Arguments

Inputs		
preserve	You can preserve the continuum or remove it	
	allowed:	bool
	Default:	True
mode	The on/off decition mode	
	allowed:	string
	Default:	'paired'
verify	Verify result (not yet implemented)	
	allowed:	bool
	Default:	False

Returns
scantable

Example

sd.scantable.auto_sinusoid_baseline.html

sd.scantable.auto_sinusoid_baseline - Function

5.1.2 Perform automatic line finding and baseline subtraction using sinusoidal function

Description

Return a scan which has been baselined (all rows) with sinusoidal functions.
Fit will be done with 'sigma-clipping'.
Spectral lines are detected first using linefinder and masked out to avoid them affecting the baseline solution.
If applyfft is set to True, the function performs Fourier analysis to select wave numbers for sinusoidal fitting. Currently, 'fft' is only available to be used for the analysis. You can specify threshold for selection of wave number using fftthresh parameter. Both float and string is acceptable. Given a float value, the unit is set to sigma. For string values, allowed formats include:

- any decimal number plus 'sigma' (e.g. '3sigma')
- 'top' plus any decimal number (e.g. 'top10')

In addition, you can add or reject specific wave numbers from the fit using addwn and rejwn, respectively. You can specify wave numbers as an integer, string, or list of them. For string specification, syntax for those parameters are as follows:

- 'a-b' (= a, a+1, a+2, ..., b-1, b)
- '<a' (= 0, 1, ..., a-2, a-1)
- '>a' (= a+1, a+2, ... up to maximum wave number corresponding to the Nyquist frequency)

You can append '=' after inequality sign. When both addwn and rejwn are set, rejwn will take priority of addwn.
The edge argument is 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).
The threshold and chan_avg_limit arguments are linefinder options. The former is the threshold used by line finder. It is better to keep it large as only strong lines affect the baseline solution. The later is a maximum number of consequtive 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. See linefinder for more details on these options.
Note: The best-fit parameter values output in logger and/or blfile are now based on specunit of 'channel'.

Arguments

Inputs	
insitu	If False a new scantable is returned
	allowed: bool
	Default: None (use default value)
mask	An optional mask retreived from scantable
	allowed: bool array
	Default: None (no mask)
applyfft	Perform Fourier analysis to find appropriate sinusoidal component
	allowed: bool
	Default: True
fftmethod	Method to find sinusoidal component (currently only 'fft' is available)
	allowed: string
	Default: 'fft'
fftthresh	Threshod to select wave number in Fourier analysis
	allowed: float, string
	Default: 3.0
addwn	Additional wave numbers to be used for fitting
	allowed: integer, string, any array
	Default: []
rejwn	Waver numbers not to be used for fitting
	allowed: integer, string, any array
	Default: []
clipthresh	Clipping threshold in unit of sigma
	allowed: float
	Default: 3.0
clipniter	Maximum number of iteration of clipping
	allowed: integer
	Default: 0
edge	An optional number of channel to drop at the edge of spectrum
	allowed: integer, integer array
	Default: (0,0)
threshold	The threshold used by line finder
	allowed: float
	Default: 3
chan_avg_limit	A maximum number of consequtive spectral channels to average during the search
	allowed: int
	Default: 1
plot	Plot the fit and the residual (currently unavailable)
	allowed: bool
	Default: False
getresidual	If False, return best-fit value instead of residual
	allowed: bool
	Default: True
showprogress	Show progress status for large data
	allowed: bool
	Default: True
minnrow	Minimum number of spectra to show progress status
	allowed: integer
	Default: 1000
outlog	Output the coefficients of the best-fit function to logger
	allowed: bool
	Default: False

Returns
scantable

Example

```
scan = sd.scantable('OrionS_rawACSmod_cal',average=False)
scan2 = scan.auto_sinusoid_baseline(addwn='<=10', insitu=False)
```

sd.scantable.average_beam.html

sd.scantable.average_beam - Function

5.1.2 Average beams together for multi-beam observation

Description

Average the Beams together.
The mask argument is an optional mask defining the region, where the averaging will be applied. The output will have all specified points masked.
The weight argument specifies weighting scheme. Valid options are:

'none':	no weight (default)
'var':	1/var(spec) weighted
'tsys':	1/Tsys**2 weighted

Arguments

Inputs		
mask	An optional mask defining the region	
	allowed:	bool array
	Default:	None
weight	Weighting scheme	
	allowed:	string
	Default:	'none'

Returns
scantable

Example

sd.scantable.average_pol.html

sd.scantable.average_pol - Function

5.1.2 Average polarizations together

Description

Average the Polarisations together.
The mask argument is an optional mask defining the region, where the averaging will be applied. The output will have all specified points masked.
The weight argument specifies weighting scheme. Valid options are:

'none':	no weight (default)
'var':	1/var(spec) weighted
'tsys':	1/Tsys**2 weighted

Arguments

Inputs		
mask	An optional mask defining the region	
	allowed:	bool array
	Default:	None
weight	Weighting scheme	
	allowed:	string
	Default:	'none'

Returns
scantable

Example

sd.scantable.average_time.html

sd.scantable.average_time - Function

5.1.2 Return time average of a scan

Description

Return the (time) weighted average of a scan.
The weight argument specifies weighting scheme. Valid options are:

'none':	no weight
'var':	1/var(spec) weighted
'tsys':	1/Tsys**2 weighted
'tint':	integration time weighted (default)
'tintsys':	Tint/Tsys**2 weightd
'median':	median averaging

The align argument is effective only for channel. If it is True, align the spectra in velocity before averaging. It takes the time of the first spectrum as reference time.

Arguments

Inputs		
mask	An optional mask (only used for 'var' and 'tsys' weighting)	
	allowed:	bool array
	Default:	None
scanav	True averages each scan separately, False averages all scans together	
	allowed:	bool
	Default:	False
weight	Weighting scheme	
	allowed:	string
	Default:	'none'
align	Align the spectral in velocity before averaging	
	allowed:	bool
	Default:	False
compel	True forces to average overwrapped IFs	
	allowed:	bool
	Default:	False

Returns

scantable

Example

```
scan = sd.scantable('OrionS_rawACSmod_cal',average=False)
# time average the scantable without using a mask
newscan = scan.average_time()
```

sd.scantable.bin.html

sd.scantable.bin - Function

5.1.2 Perform binning of spectra

Description

Return a scan where all spectra have been binned up. If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.

Arguments

Inputs		
width	The bin width in pixels	
	allowed:	integer
	Default:	5
insitu	If False a new scantable is returned	
	allowed:	bool
	Default:	None (use default value)

Returns
scantable

Example

sd.scantable.chan2data.html

sd.scantable.chan2data - Function

5.1.2 Return channel/frequency/velocity and spectral value at an arbitrary row and channel

Description

Returns channel/frequency/velocity and spectral value at an arbitrary row and channel in the scantable.
The returned value is a tuple with length of 2. The first element is a dictionary that contains an unit and a value for the abcissa, while the second one is also a dictionary that contains an unit and a value for the ordinate.

Arguments

Inputs		
rowno	A row number in the scantable	
	allowed:	integer
	Default:	0
chan	A channel in the scantable	
	allowed:	integer
	Default:	0

Returns

dictionary array

Example

```
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
s.chan2data(rowno=0,chan=0)
({'unit': 'channel', 'value': 0.0}, {'unit': 'K', 'value': 1.7028001546859741})
```

sd.scantable.clip.html

sd.scantable.clip - Function

5.1.2 Flag data by its spectral value

Description

Flag the selected data outside a specified range (in channel-base).
The method requires to set upper and lower threshold for spectral value. If clipoutside is True, the data outside the range will be flagged/unflagged. On the other hand, if it is False, the data inside the range will be flagged/unflagged. The operation if flag or unflag is controlled by the unflag argument.
Note that the operation will always be applied to this scantable regardless of the value of sd.rcParams['insitu'].

Arguments

Inputs		
uthres	Upper threshold	
	allowed:	float
	Default:	None
dthres	Lower threshold	
	allowed:	float
	Default:	None
clipoutside	True for flagging data outside the range, False for flagging data inside the range	
	allowed:	bool
	Default:	True
unflag	If True, unflag the data	
	allowed:	bool
	Default:	False

Returns

Example

sd.scantable.convert_flux.html

sd.scantable.convert_flux - Function

5.1.2 Return a scan where all spectra are converted to Jy or K

Description

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. If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.

Arguments

Inputs		
jyperk	The Jy/K conversion factor	
	allowed:	float
	Default:	None
eta	The aperture efficiency	
	allowed:	float
	Default:	None
d	The geometric diameter (metres)	
	allowed:	float
	Default:	None
insitu	If False a new scantable is returned	
	allowed:	bool
	Default:	None

Returns
scantable

Example

sd.scantable.convert_pol.html

sd.scantable.convert_pol - Function

5.1.2 Convert data to a different polarization type

Description

Convert the data to a different polarisation type. Note that you will need cross-polarisation terms for most conversions.
The poltype argument specifies the new polarization type. Valid types are: 'linear', 'circular', 'stokes', and 'linpol'.

Arguments

Inputs		
poltype	The new polarization type	
	allowed:	string
	Default:	None

Returns
scantable

Example

sd.scantable.copy.html

sd.scantable.copy - Function

5.1.2 Return a copy of this scantable

Description

Return a copy of this scantable.
Note that this makes a full (deep) copy. scan2 = scan1 makes a reference.

Arguments

Returns
scantable

Example

```
s=sd.scantable('OrionS_rawACSmod_cal',average=True)
# deep copy
copiedscan=s.copy()
# this makes a reference
s2=s
```

sd.scantable.create_mask.html

sd.scantable.create_mask - Function

5.1.2 Return a bool array based on [min,max] windows

Description

Compute and return a mask based on [min, max] windows. The specified windows are to be INCLUDED, when the mask is applied.
The mask window should be given as a pairs of start/end points, e.g. [min, max], [min2, max2], ... If the invert argument specified as True, return an inverted mask, i.e. the regions specified are EXCLUDED. The mask is created using the specified row for unit conversions. This is only necessary if frequency varies over rows.

Arguments

Inputs		
	Pairs of start/end points	
	allowed:	list or sequence of lists
	Default:	
invert	Determine the operation is inclusive or exclusive	
	allowed:	bool
	Default:	False (inclusive)
row	create the mask using the specified row for unit conversions	
	allowed:	integer
	Default:	0

Returns
bool array

Example

```
scan = sd.scantable('OrionS_rawACSmod_cal',average=True)
scan.set_unit('channel')
# a)
msk = scan.create_mask([400, 500], [800, 900])
# masks everything outside 400 and 500
# and 800 and 900 in the unit 'channel'
```

```
# b)
msk = 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
msk =  scan.create_mask([400])
```

sd.scantable.cspline_baseline.html

sd.scantable.cspline_baseline - Function

5.1.2 Perform a baseline subtraction using cubic spline function

Description

Return a scan which has been baselined (all rows) by cubic spline function (piecewise cubic polynomial). Fit will be done with 'sigma-clipping'.
Note: The best-fit parameter values output in logger and/or blfile are now based on specunit of 'channel'.

Arguments

Inputs	
insitu	If False a new scantable is returned allowed: bool Default: None (use default value)
mask	An optional mask retreived from scantable allowed: bool Default: None (no mask)
npiece	Number of pieces allowed: integer Default: 2
clipthresh	Clipping threshold in unit of sigma allowed: float Default: 3.0
clipniter	Maximum number of iteration of clipping allowed: integer Default: 0
plot	Plot the fit and the residual (currently unavailable) allowed: bool Default: False
getresidual	If False, return best-fit value instead of residual allowed: bool Default: True
showprogress	Show progress status for large data allowed: bool Default: True
minnrow	Minimum number of spectra to show progress status allowed: integer Default: 1000
outlog	Output the coefficients of the best-fit function to logger allowed: bool Default: False
blfile	Name of text file in which the best-fit parameter values to be written allowed: string Default: ”

Returns
scantable

Example

```
scan = sd.scantable('OrionS_rawACSmod_cal',average=False)
# return a scan baselined by a cubic spline consisting of 2 pieces (i.e., 1 internal
# also with 3-sigma clipping, iteration up to 4 times
bscan = scan.cspline_baseline(npiece=2,clipthresh=3.0,clipniter=4)
```

sd.scantable.drop_scan.html

sd.scantable.drop_scan - Function

5.1.2 Return a new scantable where the specified scan number(s) are dropped

Description

Return a new scantable where the specified scan number(s) has(have) been dropped.
It always returns a new scantable regardless of sd.rcParams['insitu'].

Arguments

Inputs		
scanid	A (list of) scan number(s)	
	allowed:	integer, integer array
	Default:	None

Returns
scantable

Example

sd.scantable.fft.html

sd.scantable.fft - Function

5.1.2 Apply FFT to the spectra

Description

Apply FFT to the spectra.
Flagged data in the scantable get interpolated over the region. An optional channel mask specified by mask parameter is applied to all specified rows if it is given as one dimensional array.
It returns a list of dictionaries containing the results for each spectrum. Each dictionary contains two values, the real and the imaginary parts when getrealimag = True, or the amplitude(absolute value) and the phase(argument) when getrealimag = False. The key for these values are 'real' and 'imag', or 'ampl' and 'phase', respectively.

Arguments

Inputs		
rowno	Row number(s) to be processed.	
	allowed:	integet, integer list or tuple
	Default:	[] (apply to whole data)
mask	An optional channel mask	
	allowed:	bool array
	Default:	[]
getrealimag	If True, return real and imaginary part instead of amplitude and phase	
	allowed:	bool
	Default:	False (return amplitude and phase)

Returns
dictionary

Example

sd.scantable.flag.html

sd.scantable.flag - Function

5.1.2 Flag selected data using specified mask (channel based flag)

Description

Flag the selected data using an optional channel mask. The appropriate value of the mask can be created with create_mask. If no mask is specified, all channels are flagged/unflagged depending on the unflag argument.
Note that the operation will always be applied to this scantable regardless of the value of sd.rcParams['insitu'].

Arguments

Inputs		
mask	An optional channel mask	
	allowed:	bool array
	Default:	None (all channels)
unflag	If True, unflag the data	
	allowed:	bool
	Default:	False

Returns

Example

sd.scantable.flag_nans.html

sd.scantable.flag_nans - Function

5.1.2 Flag NaN values

Description

Utility function to flag NaN values in the scantable.
Note that the operation will always be applied to this scantable regardless of the value of sd.rcParams['insitu'].

Arguments

Returns

Example

sd.scantable.flag_row.html

sd.scantable.flag_row - Function

5.1.2 Flag spectra based on specified rows (row based flag)

Description

Flag the selected data in row-based manner.
Note that the operation will always be applied to this scantable regardless of the value of sd.rcParams['insitu'].

Arguments

Inputs		
rows	List of row numbers to be flagged	
	allowed:	integer, integer array
	Default:	[]
unflag	If True, unflag the data	
	allowed:	bool
	Default:	False

Returns

Example

sd.scantable.freq_align.html

sd.scantable.freq_align - Function

5.1.2 Perform frequency alignment

Description

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. The reference time to align can be specified as reftime argument. By default, the time of the first row of data is used. The method argument specifies interpolation method for regridding the spectra. Valid options are 'nearest', 'linear', 'cubic' (default), and 'spline'.

Arguments

Inputs		
reftime	Reference time to align at.	
	allowed:	string
	Default:	None (use first row of data)
method	Interpolation method for regridding the spectra	
	allowed:	string
	Default:	'cubic'
insitu	If False a new scantable is returned	
	allowed:	bool
	Default:	None

Returns
scantable

Example

sd.scantable.freq_switch.html

sd.scantable.freq_switch - Function

5.1.2 Apply frequency switching to the data

Description

Apply frequency switching to the data.

Arguments

Inputs		
insitu	If False a new scantable is returned	
	allowed:	bool
	Default:	None

Returns
scantable

Example

sd.scantable.gain_el.html

sd.scantable.gain_el - Function

5.1.2 Apply gain-elevation correction based on user-provided correction factors

Description

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.
The polynomial coefficients to compute a gain-elevation correction should be given as that of a function of elevation in degrees. Contents of the ascii table specified by filename argument is correction factors as a function of time and elevation (in degree). 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

The interpolation method can be specified by method argument. Valid options are 'nearest', 'linear' (default), 'cubic', and 'spline'.

Arguments

Inputs		
poly	Polynomial coefficients to comput a gain-elevation correction	
	allowed:	float array
	Default:	None
filename	The name of an ascii file holding correction factors	
	allowed:	string
	Default:	”
method	Interpolation method when correcting from a table	
	allowed:	string
	Default:	’linear’
insitu	If False a new scantable is returned	
	allowed:	bool
	Default:	None

Returns
scantable

Example

sd.scantable.get_abcissa.html

sd.scantable.get_abcissa - Function

5.1.2 Get the abcissa values and format string that represents current coordinate setup

Description

Get the abcissa in the current coordinate setup for the currently selected Beam/IF/Pol. The method returns the abcissa values and the format string as a dictionary.

Arguments

Inputs		
rowno	An optional row number in the scantable	
	allowed:	integer
	Default:	0

Returns
dictionary

Example

sd.scantable.get_antennaname.html

sd.scantable.get_antennaname - Function

5.1.2 Get antenna name

Description

Return a name of antenna observed.

Arguments

Returns
string

Example

sd.scantable.get_azimuth.html

sd.scantable.get_azimuth - Function

5.1.2 Get a list of azimuth during the observation

Description

Get a list of azimuths for the observations. Return a float for each integration in the scantable. The unit is radian.

Arguments

Inputs		
row	Row no of integration	
	allowed:	integer
	Default:	-1 (all rows)

Returns
float, float array

Example

sd.scantable.get_column_names.html

sd.scantable.get_column_names - Function

5.1.2 Get a list of column names in the main table

Description

Return a list of column names in the main table, which can be used for selection.

Arguments

Returns
string array

Example

sd.scantable.get_coordinate.html

sd.scantable.get_coordinate - Function

5.1.2 Return the spectral coordinate for a given row as a coordinate object

Description

Return the (spectral) coordinate for a a given 'rowno'.
Notes:

- This coordinate is only valid until a scantable method modifies the frequency axis.
- This coordinate does contain the original frequency set-up NOT the new frame. The conversions however are done using the user specified frame (e.g. LSRK/TOPO). To get the 'real' coordinate, use scantable.freq_align first. Without it there is no closure, i.e.::

```
c = myscan.get_coordinate(0)
c.to_frequency(c.get_reference_pixel()) != c.get_reference_value()
```

Arguments

Inputs		
rowno	The row number for the spectral coordinate	
	allowed:	integer
	Default:	

Returns
coordinate

Example

sd.scantable.get_direction.html

sd.scantable.get_direction - Function

5.1.2 Get a list of positions on the sky as a string

Description

Get a list of Positions on the sky (direction) for the observations. Return a string for each integration in the scantable.
To get float value for positions, use get_directionval.

Arguments

Inputs		
row	Row no of integration	
	allowed:	integer
	Default:	-1 (all rows)

Returns

string, string array

Example

```
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
# get_direction() returns string expression of position
s.get_direction(0)
'05:35:13.5 -05.24.08.2'
# s.get_directionval() returns float value of position
s.get_directionval(0)
[1.4626913601468896, -0.0942875343295448]
```

sd.scantable.get_directionval.html

sd.scantable.get_directionval - Function

5.1.2 Get a list of positions on the sky as a float

Description

Get a list of Positions on the sky (direction) for the observations. Return a float for each integration in the scantable. The unit is radian.
To get string expression for positions, use get_direction.

Arguments

Inputs		
row	Row no of integration	
	allowed:	integer
	Default:	-1 (all rows)

Returns

float array

Example

```
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
# get_direction() returns string expression of position
s.get_direction(0)
'05:35:13.5 -05.24.08.2'
# s.get_directionval() returns float value of position
s.get_directionval(0)
[1.4626913601468896, -0.0942875343295448]
```

sd.scantable.get_elevation.html

sd.scantable.get_elevation - Function

5.1.2 Get a list of elevation during the observation

Description

Get a list of elevations for the observations. Return a float for each integration in the scantable. The unit is radian.

Arguments

Inputs		
row	Row no of integration	
	allowed:	integer
	Default:	-1 (all rows)

Returns
float, float array

Example

sd.scantable.get_fit.html

sd.scantable.get_fit - Function

5.1.2 Get the stored fits for a row in scantable

Description

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

Arguments

Inputs		
row	The row which the fit has been applied to	
	allowed:	integer
	Default:	0

Returns
dictionary

Example

sd.scantable.get_fluxunit.html

sd.scantable.get_fluxunit - Function

5.1.2 Get a flux unit

Description

Return a flux unit string.

Arguments

Returns
string

Example

sd.scantable.get_inttime.html

sd.scantable.get_inttime - Function

5.1.2 Get a list of integration times for the observation

Description

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

Arguments

Inputs		
row	The row which the fit has been applied to	
	allowed:	integer
	Default:	-1 (all rows)

Returns
float, float array

Example

sd.scantable.get_mask.html

sd.scantable.get_mask - Function

5.1.2 Get mask for the specified row as a bool list

Description

Return the mask for the current row in the scantable as a list.

Arguments

Inputs		
rowno	The row number to retrieve the mask from	
	allowed:	integer
	Default:	

Returns
bool array

Example

sd.scantable.get_mask_indices.html

sd.scantable.get_mask_indices - Function

5.1.2 Compute and return lists of mask start indices and mask end indices from the given bool array

Description

Compute and Return lists of mask start indices and mask end indices.
Returned value is a list of mask start indices and that of mask end indices, i.e., ([istart1,istart2,....], [iend1,iend2,....]).

Arguments

Inputs		
mask	Channel mask, created with create_mask	
	allowed:	bool array
	Default:	None

Returns

integer array

Example

```
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
# this is an example to show how get_mask_indices() works
s.get_mask_indices(s.create_mask([1000,2000],[4000,5000]))
([1000, 4000], [2000, 5000])
```

sd.scantable.get_masklist.html

sd.scantable.get_masklist - Function

5.1.2 Compute and return a list of mask windows [min,max] from the given bool array

Description

Compute and return a list of mask windows, [min, max]. Returned value is pairs of start/end points (inclusive) specifying the masked regions, i.e. [min, max], [min2, max2], ...
The row argument specifies the row to use for unit conversions, default is row=0. It is only necessary if frequency varies over rows.

Arguments

Inputs		
mask	Channel mask, created with create_mask	
	allowed:	bool array
	Default:	None
row	calculate the masklist using the specified row for unit conversion	
	allowed:	integer
	Default:	0
silent	True for silent mode	
	allowed:	bool
	Default:	False

Returns
float array

Example

```
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
# this is an example to show how get_masklist() works
s.get_masklist(s.create_mask([1000,2000],[4000,5000]))
([1000.0, 2000.0], [4000.0, 5000.0])
```

sd.scantable.get_parangle.html

sd.scantable.get_parangle - Function

5.1.2 Get a list of parallactic angles for the observation

Description

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

Arguments

Inputs		
row	Row no of integration	
	allowed:	integer
	Default:	-1 (all rows)

Returns
float, float array

Example

sd.scantable.get_restfreqs.html

sd.scantable.get_restfreqs - Function

5.1.2 Get the rest frequency(s) stored in the scantable

Description

Get the restfrequency(s) stored in this scantable. The return value(s) are always of unit 'Hz' The method returns a dictionary containing ids and a list of doubles for each id.
The rest frequency is stored in MOLECULES subtable of the scantable. Rows in MOLECULES subtable is referred from main table by MOLECULE_ID. You can specify a list of that id using ids argument to retrieve particular rest frequency(s).

Arguments

Inputs		
ids	A list of MOLECULE_ID for that rest frequency(s) to be retrieved	
	allowed:	integer
	Default:	None (retrieve all)

Returns
dictionary

Example

sd.scantable.get_rms.html

sd.scantable.get_rms - Function

5.1.2 Calculate rms of the spectrum

Description

Calculate rms of the spectrum. Mask can be specified.

Arguments

Inputs		
mask	Optional mask for calculation	
	allowed:	bool array
	Default:	
whichrow	Row number to be processed	
	allowed:	int
	Default:	

Returns
float

Example

sd.scantable.get_row.html

sd.scantable.get_row - Function

5.1.2 Return a scantable with single row

Description

Select a row in the scantable. Return a scantable with single row.
If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.

Arguments

Inputs		
row	Row no of integration	
	allowed:	integer
	Default:	0
insitu	If False a new scantable is returned	
	allowed:	bool
	Default:	None

Returns
scantable

Example

sd.scantable.get_row_selector.html

sd.scantable.get_row_selector - Function

5.1.2 Return a selector object that only selects target row

Description

Return a selector object that only selects target row.

Arguments

Inputs		
rowno	The row number to select	
	allowed:	integer
	Default:	

Returns
selector

Example

sd.scantable.get_scan.html

sd.scantable.get_scan - Function

5.1.2 Return a specified scan(s) specified by scan number or source name as a new scantable

Description

Return a specific scan (by scanno) or collection of scans (by source name) in a new scantable. In both case, you can set selection criteria to scanid argument. For source name, unix-style patterns are accepted for source name matching, e.g. '*_R' gets all 'ref scans.
Note that drop_scan is an inverse operation.

Arguments

Inputs		
scanid	A (list of) scanno or a source name	
	allowed:	integer, integer array, string
	Default:	None

Returns
scantable

Example

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

sd.scantable.get_selection.html

sd.scantable.get_selection - Function

5.1.2 Get current selection that is currently set on this scantable

Description

Get the selection object currently set on this scantable.

Arguments

Returns
selector

Example

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

sd.scantable.get_sourcename.html

sd.scantable.get_sourcename - Function

5.1.2 Get a list of source names for the observation

Description

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

Arguments

Inputs		
row	Row no of integration	
	allowed:	integer
	Default:	-1 (all rows)

Returns
string, string array

Example

sd.scantable.get_spectrum.html

sd.scantable.get_spectrum - Function

5.1.2 Get the spectrum for the current row

Description

Return the spectrum for the current row in the scantable as a list.

Arguments

Inputs		
rowno	The row number to retrieve the spectrum from	
	allowed:	integer
	Default:	

Returns
float array

Example

sd.scantable.get_time.html

sd.scantable.get_time - Function

5.1.2 Get a list of time stamps for the observation

Description

Get a list of time stamps for the observations. Return a datetime object or a string (default) for each integration time stamp in the scantable.

Arguments

Inputs		
row	Row no of integration	
	allowed:	integer
	Default:	-1 (all rows)
asdatetime	Return values as datetime objects rather than strings	
	allowed:	bool
	Default:	False

Returns

string, datetime, their array

Example

sd.scantable.get_tsys.html

sd.scantable.get_tsys - Function

5.1.2 Get a list of system temperatures

Description

Return the System temperatures.

Arguments

Inputs		
row	The rowno to get the information for	
	allowed:	integer
	Default:	-1 (all rows)

Returns
float array

Example

sd.scantable.get_unit.html

sd.scantable.get_unit - Function

5.1.2 Get the default unit of spectral axis

Description

Get the default unit set for spectral axis in this scantable

Arguments

Returns
string

Example

sd.scantable.get_weather.html

sd.scantable.get_weather - Function

5.1.2 Get the weather informations

Description

Return the weather informations.
The contents of returned dictionary is as follows:

humidity:	relative humidity
pressure:	atmospheric pressure
temperature:	atmospheric temperature
windaz:	wind direction in radian
windspeed:	wind speed in m/s

Arguments

Inputs		
row	The rowno to get the information for	
	allowed:	integer
	Default:	-1 (all rows)

Returns
dictionary, dictionary array

Example

sd.scantable.getbeam.html

sd.scantable.getbeam - Function

5.1.2 Get beam number of the given row

Description

Return a beam number for the given row.

Arguments

Inputs	
	The rowno to get the information for
	allowed: integer
	Default:

Returns

integer

Example

```
scan = sd.scantable('OrionS_rawACSmod')
beam = scan.getbeam(0) # get beam number for the first row
```

sd.scantable.getbeamnos.html

sd.scantable.getbeamnos - Function

5.1.2 Get a list of beam numbers in the scantable

Description

Return a list of beam numbers in the scantable.

Arguments

Returns
integer array

Example

sd.scantable.getcycle.html

sd.scantable.getcycle - Function

5.1.2 Get cycle number of the given row

Description

Return a cycle number for the given row.

Arguments

Inputs	
	The rowno to get the information for
	allowed: integer
	Default:

Returns
integer

Example

```
scan = sd.scantable('OrionS_rawACSmod')
cycle = scan.getcycle(0) # get cycle number for the first row
```

sd.scantable.getif.html

sd.scantable.getif - Function

5.1.2 Get IF number of the given row

Description

Return a IF number for the given row.

Arguments

Inputs	
	The rowno to get the information for
	allowed: integer
	Default:

Returns
integer

Example

```
scan = sd.scantable('OrionS_rawACSmod')
ifno = scan.getifs(0) # get IF number for the first row
```

sd.scantable.getifnos.html

sd.scantable.getifnos - Function

5.1.2 Get a list of IF nubers in the scantable

Description

Return a list of IF numbers in the scantable.

Arguments

Returns
integer array

Example

sd.scantable.getmolnos.html

sd.scantable.getmolnos - Function

5.1.2 Get a list of molecule ids in the scantable

Description

Return a list of molecule ids in the scantable.

Arguments

Returns
integer array

Example

sd.scantable.getpol.html

sd.scantable.getpol - Function

5.1.2 Get polarization number of the given row

Description

Return a polarization number for the given row.

Arguments

Inputs		
	The rowno to get the information for	
	allowed:	integer
	Default:	

Returns
integer

Example

```
scan = sd.scantable('OrionS_rawACSmod')
polno = scan.getpol(0) # get polarization number for the first row
```

sd.scantable.getpolnos.html

sd.scantable.getpolnos - Function

5.1.2 Get a list of polarization numbers in the scantable

Description

Return a list of polarization numbers in the scantable.

Arguments

Returns
integer array

Example

sd.scantable.getscan.html

sd.scantable.getscan - Function

5.1.2 Get scan number of the given row

Description

Return a scan number for the given row.

Arguments

Inputs		
	The rowno to get the information for	
	allowed:	integer
	Default:	

Returns
integer

Example

```
scan = sd.scantable('OrionS_rawACSmod')
scanno = scan.getscan(0) # get scan number for the first row
```

sd.scantable.getscannos.html

sd.scantable.getscannos - Function

5.1.2 Get a list of scan numbers in the scantable

Description

Return a list of scan numbers in the scantable.

Arguments

Returns
integer array

Example

sd.scantable.history.html

sd.scantable.history - Function

5.1.2 Print a history

Description

Print the history. Optionally to a file.

Arguments

Inputs		
filename	The name of the file to save the history to	
	allowed:	string
	Default:	None

Returns

string

Example

sd.scantable.invert_phase.html

sd.scantable.invert_phase - Function

5.1.2 Invert the phase of the complex polarization

Description

Invert the phase of the complex polarisation.

Arguments

Returns

Example

sd.scantable.lag_flag.html

sd.scantable.lag_flag - Function

5.1.2 Perform Fourier filtering on the spectra

Description

Flag the data in 'lag' space by providing a frequency to remove. Flagged data in the scantable gets interpolated over the region. No taper is applied.
If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.
It is recommended to flag edges of the band or strong signals beforehand.

Arguments

Inputs		
start	The start frequency (really a period within the bandwidth) or period to remove	
	allowed:	float
	Default:	
end	The end frequency or period to remove	
	allowed:	string
	Default:	None
unit	The frequency unit or " for explicit lag channels	
	allowed:	string
	Default:	'MHz'
insitu	If False a new scantable is returned	
	allowed:	bool
	Default:	None

Returns
scantable

Example

sd.scantable.mx_quotient.html

sd.scantable.mx_quotient - Function

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

Description

Form a quotient using "off" beams when observing in "MX" mode.
The formula to get result depends on the preserve paramter. If it is True, the continuum will be preserved while if it is False, the continuum will be removed. The equation used are

$$T_{\rm a}^{*} = T_{\rm sys}^{\rm OFF}\frac{ON}{OFF} - T_{\rm sys}^{\rm OFF},$$

if preserve is True, while

$$T_{\rm a}^{*} = T_{\rm sys}^{\rm OFF}\frac{ON}{OFF} - T_{\rm sys}^{\rm ON},$$

if preserve is False.
The weight argument is used for time averaging off beams. You can set any options same as average_time.

Arguments

Inputs		
mask	An optional mask to be used when weight == 'stddev'	
	allowed:	bool array
	Default:	None
weight	How to average the off beams	
	allowed:	string
	Default:	'median'
preserve	Preserve the continuum or remove it	
	allowed:	bool
	Default:	True

Returns
scantable

Example

sd.scantable.nbeam.html

sd.scantable.nbeam - Function

5.1.2 Return a number of beams

Description

Return a number of beams in this scantable.
If scanno argument is specified, the number of beams for that scan will be returned. Otherwise, the total number of beams, which is written in the header of the data, will be returned.

Arguments

Inputs		
scanno	Scan number	
	allowed:	integer
	Default:	-1 (total)

Returns
integer

Example

sd.scantable.nchan.html

sd.scantable.nchan - Function

5.1.2 Return a number of channels

Description

Return a number of channels in this scantable.
If ifno argument is specified, the number of channels for that IF will be returned. Otherwise, the maximum number of channels, which is written in the header of the data, will be returned.

Arguments

Inputs		
ifno	IF number	
	allowed:	integer
	Default:	-1 (maximum)

Returns
integer

Example

sd.scantable.ncycle.html

sd.scantable.ncycle - Function

5.1.2 Return a number of cycles

Description

Return a number of cycles in this scantable.
If scanno argument is specified, the number of cycles for that scan will be returned. Otherwise, the total number of cycles will be returned.

Arguments

Inputs		
scanno	Scan number	
	allowed:	integer
	Default:	-1 (total)

Returns
integer

Example

sd.scantable.nif.html

sd.scantable.nif - Function

5.1.2 Return a number of IFs

Description

Return a number of IFs in this scantable.
If scanno argument is specified, the number of IFs for that scan will be returned. Otherwise, the total number of IFs, which is written in the header of the data, will be returned.

Arguments

Inputs		
scanno	Scan number	
	allowed:	integer
	Default:	-1 (total)

Returns
integer

Example

sd.scantable.npol.html

sd.scantable.npol - Function

5.1.2 Return a number of polarizations

Description

Return a number of polarizations in this scantable.
If scanno argument is specified, the number of polarizations for that scan will be returned. Otherwise, the total number of polarizations, which is written in the header of the data, will be returned.

Arguments

Inputs		
scanno	Scan number	
	allowed:	integer
	Default:	-1 (total)

Returns
integer

Example

sd.scantable.nrow.html

sd.scantable.nrow - Function

5.1.2 Return a number of rows

Description

Return a total number of rows (= total number of spectra) in this scantable.

Arguments

Returns
integer

Example

sd.scantable.nscan.html

sd.scantable.nscan - Function

5.1.2 Return a number of scans

Description

Return a total number of scans in this scantable.

Arguments

Returns
integer

Example

sd.scantable.opacity.html

sd.scantable.opacity - Function

5.1.2 Apply an opacity correction

Description

Apply an opacity correction. The data and Tsys are multiplied by the correction factor.
Correction includes an elevation dependence of the opacity. Actual correction factor is $\exp(\tau * \mathrm{ZD})$, where ZD is the zenith-distance and τ is a value given as the tau argument. If a list of opacities is provided, it has to be of length nIF, nIF*nPol or 1 and in order of IF/POL, e.g. [opif0pol0, opif0pol1, opif1pol0 ...].
If tau is 'None' the opacities are determined from a model.
If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.

Arguments

Inputs		
tau	A (list of) opacity	
	allowed:	float, float array
	Default:	None
insitu	If False a new scantable is returned	
	allowed:	bool
	Default:	None

Returns
scantable

Example

sd.scantable.parallactify.html

sd.scantable.parallactify - Function

5.1.2 Set a flag to indicate the data should be treated as "parallactified"

Description

Set a flag to indicate whether this data should be treated as having been 'parallactified' (total phase == 0.0)

Arguments

Inputs	
pflag	Bool indicating whether to turn this on (True) or off (False) allowed: bool Default:

Returns

Example

sd.scantable.poltype.html

sd.scantable.poltype - Function

5.1.2 Get a plarization type

Description

Return a polarization type in this scantable.

Arguments

Returns
string

Example

sd.scantable.poly_baseline.html

sd.scantable.poly_baseline - Function

5.1.2 Perform a baseline subtraction using polynomial function

Description

Return a scan which has been baselined (all rows) by a polynomial.
If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.
You can verify and decide whether you apply the fit result or not, if plot argument is True. In that case, you have to answer 'y' or 'n' for each spectra so that setting True is not recommended for large dataset.
This is an newer version of old_poly_baseline. The only difference between them is its performance. For larger dataset, it is recommended to use this method.

Arguments

Inputs		
mask	An optional mask	
	allowed:	bool array
	Default:	None
order	The order of the polynomial	
	allowed:	integer
	Default:	0
insitu	If False a new scantable is returned	
	allowed:	bool
	Default:	None (use default value)
plot	Plot the fit and the residual	
	allowed:	bool
	Default:	False
getresidual	If False, return best-fit value instead of residual	
	allowed:	bool
	Default:	True
showprogress	Show progress status for large data	
	allowed:	bool
	Default:	True
minnrow	Minimum number of spectra to show progress status	
	allowed:	integer
	Default:	1000
outlog	Output the coefficients of the best-fit function to logger	
	allowed:	bool
	Default:	False
blfile	Name of text file in which the best-fit parameter values to be written	
	allowed:	string
	Default:	”

Returns
scantable

Example

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

sd.scantable.recalc_azel.html

sd.scantable.recalc_azel - Function

5.1.2 Recalculate azimuth and elevation for each sky position

Description

Recalculate the azimuth and elevation for each position.
Note that the operation will always be applied to this scantable regardless of the value of sd.rcParams['insitu'].

Arguments

Returns

Example

sd.scantable.resample.html

sd.scantable.resample - Function

5.1.2 Perform a binning

Description

Return a scan where all spectra have been binned up.
The method argument specifies interpolation method when correcting from a table. Values are 'nearest', 'linear', 'cubic' (default), and 'spline'.
If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.

Arguments

Inputs		
width	The bin width in pixels	
	allowed:	integer
	Default:	5
method	Interpolation method when correcting from a table	
	allowed:	string
	Default:	'cubic'
insitu	If False a new scantable is returned	
	allowed:	bool
	Default:	None

Returns
scantable

Example

sd.scantable.rotate_linpolphase.html

sd.scantable.rotate_linpolphase - Function

5.1.2 Rotate a phase of the complex polarization

Description

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.

Arguments

Inputs		
angle	The angle (in dgree) to rotate (add) by	
	allowed:	float
	Default:	

Returns

Example

```
scan = sd.scantable('OrionS_rawACSmod_cal')
scan.rotate_linpolphase(2.3)
```

sd.scantable.rotate_xyphase.html

sd.scantable.rotate_xyphase - Function

5.1.2 Rotate a phase of the XY correlation

Description

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.

Arguments

Inputs		
angle	The angle (in dgree) to rotate (add) by	
	allowed:	float
	Default:	

Returns

Example

```
scan = sd.scantable('OrionS_rawACSmod_cal')
scan.rotate_xyphase(2.3)
```

sd.scantable.save.html

sd.scantable.save - Function

5.1.2 Store the scantable on disk

Description

Store the scantable on disk. This can be an asap (CASA Table), SDFITS or MS2 format.
The output filename can be specified as name. For format 'ASCII', this is the root file name (data in 'name'.txt and header in 'name'_header.txt). If no name is given, the default name will be used. The format is optinal file format. Default is 'ASAP'. Allows are:

- 'ASAP' (save as ASAP [CASA Table])
- 'SDFITS' (save as SDFITS file)
- 'ASCII' (saves as ascii text file)
- 'MS2' (saves as an casacore MeasurementSet V2)
- 'FITS' (save as image FITS - not readable by class)
- 'CLASS' (save as FITS readable by CLASS)

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

Arguments

Inputs		
name	The name of the output file	
	allowed:	string
	Default:	None
format	An optional file format	
	allowed:	string
	Default:	None
overwrite	Overwrite existing file	
	allowed:	bool
	Default:	False

Returns

Example

```
scan = sd.scantable('OrionS_rawACSmod_cal')
scan.save('myscan.asap')
scan.save('myscan.sdfits', 'SDFITS')
```

sd.scantable.scale.html

sd.scantable.scale - Function

5.1.2 Scale spectra by the given factor

Description

Return a scan where all spectra are scaled by the given 'factor'. The factor can be a float or one- or two-dimensional float array. If it is one-dimensional array, the scaling will be done in channel-by-channel manner. If it is two-dimensional, the scaling will be done in element-by-element or row-by-row manner.
If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.

Arguments

Inputs		
factor	The scaling factor	
	allowed:	float, float array
	Default:	
insitu	If False a new scantable is returned	
	allowed:	bool
	Default:	None
tsys	If True then apply the operation to Tsys as well as the data	
	allowed:	bool
	Default:	True

Returns
scantable

Example

sd.scantable.set_dirframe.html

sd.scantable.set_dirframe - Function

5.1.2 Set the frame type of the direction on the sky

Description

Set the frame type of the Direction on the sky. The valid frames are 'J2000', 'B1950', 'GALACTIC'.

Arguments

Inputs		
frame	An optional frame type	
	allowed:	string
	Default:	"

Returns

Example

```
scan = sd.scantable('OrionS_rawACSmod_cal')
scan.set_dirframe('GALAGTIC')
```

sd.scantable.set_doppler.html

sd.scantable.set_doppler - Function

5.1.2 Set definition of the Doppler correction

Description

Set the doppler for all following operations on this scantable.

Arguments

Inputs		
doppler	One of 'RADIO', 'OPTICAL', 'Z', 'BETA', 'GAMMA'	
	allowed:	string
	Default:	'RADIO'

Returns

Example

sd.scantable.set_feedtype.html

sd.scantable.set_feedtype - Function

5.1.2 Set the feed type

Description

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

Arguments

Inputs		
feedtype	'linear' or 'circular'	
	allowed:	string
	Default:	

Returns

Example

sd.scantable.set_fluxunit.html

sd.scantable.set_fluxunit - Function

5.1.2 Set flux unit

Description

Set flux unit to this scantable. Valid values are 'K' and 'Jy'.

Arguments

Inputs		
	'K' or 'Jy'	
	allowed:	string
	Default:	

Returns

Example

sd.scantable.set_freqframe.html

sd.scantable.set_freqframe - Function

5.1.2 Set the frame type of the spectral axis

Description

Set the frame type of the Spectral Axis. Valid frames are 'TOPO', 'LSRD', 'LSRK', 'BARY', 'GEO', 'GALACTO', 'LGROUP', 'CMB'. The default is taken from sd.rcParams['scantable.freqframe'].

Arguments

Inputs		
frame	An optional frame type	
	allowed:	string
	Default:	None

Returns

Example

```
scan = sd.scantable('OrionS_rawACSmod_cal')
scan.set_freqframe('BARY')
```

sd.scantable.set_instrument.html

sd.scantable.set_instrument - Function

5.1.2 Set antenna name

Description

Set the instrument (antenna name) for subsequent processing.

Arguments

Inputs	
instr	Instrument (or antenna) name
	allowed: string
	Default:

Returns

Example

sd.scantable.set_restfreqs.html

sd.scantable.set_restfreqs - Function

5.1.2 Set rest frequency

Description

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.
Note that, to do more sophisticate Restfrequency setting, e.g. on a source and IF basis, use scantable.set_selection() before using this function::

```
# provided your scantable is called scan
selection = sd.selector()
selection.set_name("ORION*")
selection.set_ifs([1])
scan.set_selection(selection)
scan.set_restfreqs(freqs=86.6e9)
```

Arguments

Inputs		
freqs	List of rest frequency values or string identifiers	
	allowed:	string
	Default:	None
unit	Unit for rest frequency	
	allowed:	string
	Default:	'Hz'

Returns

Example

```
scan = sd.scantable('OrionS_rawACSmod_cal')
# set the given restfrequency for the all currently selected IFs
scan.set_restfreqs(freqs=1.4e9)
# set restfrequencies for the n IFs  (n > 1) in the order of the
# list, i.e
# IF0 -> 1.4e9, IF1 ->  1.41e9, IF3 -> 1.42e9
# len(list_of_restfreqs) == nIF
# for nIF == 1 the following will set multiple restfrequency for
# that IF
scan.set_restfreqs(freqs=[1.4e9, 1.41e9, 1.42e9])
# set multiple restfrequencies per IF. as a list of lists where
# the outer list has nIF elements, the inner s arbitrary
scan.set_restfreqs(freqs=[[1.4e9, 1.41e9], [1.67e9]])
```

sd.scantable.set_selection.html

sd.scantable.set_selection - Function

5.1.2 Select a subset of the data

Description

Select a subset of the data. All following operations on this scantable are only applied to the selection.
The selection can be done via a selector object (default unset the selection), or any combination of "pols", "ifs", "beams", "scans", "cycles", "name", "query", "types" that will be passed to the constructor of the selector object to create a new selection.

Arguments

Inputs		
selection	A selector object	
	allowed:	selector
	Default:	None

Returns

Example

```
scan = sd.scantable('OrionS_rawACSmod_cal')
sel = sd.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
# or the equivalent
scan.set_selection(scans=[0,3])
scan.summary()           # will only print summary of scanno 0 an 3
scan.set_selection()     # unset the selection
```

sd.scantable.set_sourcetype.html

sd.scantable.set_sourcetype - Function

5.1.2 Set the types of source to be source or reference scan

Description

Set the type of the source to be an source or reference scan using the provided pattern.

Arguments

Inputs	
match	A Unix style pattern, regular expression or selector
	allowed: string, selector
	Default:
matchtype	'pattern' for UNIX style pattern, 'regex' for regular expression
	allowed: string
	Default: 'pattern'
sourcetype	The type of the source to use (source/reference)
	allowed: string
	Default: 'reference'

Returns

Example

sd.scantable.set_spectrum.html

sd.scantable.set_spectrum - Function

5.1.2 Set spectrum for specified row

Description

Set the spectrum for the current row in the scantable.

Arguments

Inputs		
spec	The new spectrum	
	allowed:	float array
	Default:	
rowno	The row number to set the spectrum for	
	allowed:	integer
	Default:	

Returns

Example

sd.scantable.set_unit.html

sd.scantable.set_unit - Function

5.1.2 Set unit for spectral axis

Description

Set the unit for all following operations on this scantable. Valid options are '*Hz', 'km/s', 'channel', or ''.

Arguments

Inputs		
unit	Optional unit	
	allowed:	string
	Default:	'channel'

Returns

Example

sd.scantable.shift_refpix.html

sd.scantable.shift_refpix - Function

5.1.2 Shift the reference pixel of the spectral coordinate

Description

Shift the reference pixel of the Spectra Coordinate by an integer amount.
Be careful using this with broadband data.

Arguments

Inputs		
delta	The amount to shift by	
	allowed:	integer
	Default:	

Returns

Example

sd.scantable.sinusoid_baseline.html

sd.scantable.sinusoid_baseline - Function

5.1.2 Perform a baseline subtraction using sinusoidal function

Description

Return a scan which has been baselined (all rows) with sinusoidal functions.
Fit will be done with 'sigma-clipping'.
Spectral lines are detected first using linefinder and masked out to avoid them affecting the baseline solution.
If applyfft is set to True, the function performs Fourier analysis to select wave numbers for sinusoidal fitting. Currently, 'fft' is only available to be used for the analysis. You can specify threshold for selection of wave number using fftthresh parameter. Both float and string is acceptable. Given a float value, the unit is set to sigma. For string values, allowed formats include:

- any decimal number plus 'sigma' (e.g. '3sigma')
- 'top' plus any decimal number (e.g. 'top10')

In addition, you can add or reject specific wave numbers from the fit using addwn and rejwn, respectively. You can specify wave numbers as an integer, string, or list of them. For string specification, syntax for those parameters are as follows:

- 'a-b' (= a, a+1, a+2, ..., b-1, b)
- '<a' (= 0, 1, ..., a-2, a-1)
- '>a' (= a+1, a+2, ... up to maximum wave number corresponding to the Nyquist frequency)

You can append '=' after inequality sign. When both addwn and rejwn are set, rejwn will take priority of addwn.
Note: The best-fit parameter values output in logger and/or blfile are now based on specunit of 'channel'.

Arguments

Inputs		
insitu	If False a new scantable is returned	
	allowed:	bool
	Default:	None (use default value)
mask	An optional mask retreived from scantable	
	allowed:	bool array
	Default:	None (no mask)
applyfft	Perform Fourier analysis to find appropriate sinusoidal component	
	allowed:	bool
	Default:	True
fftmethod	Method to find sinusoidal component (currently only 'fft' is available)	
	allowed:	string
	Default:	'fft'
fftthresh	Threshod to select wave number in Fourier analysis	
	allowed:	float, string
	Default:	3.0
addwn	Additional wave numbers to be used for fitting	
	allowed:	integer, string, any array
	Default:	[]
rejwn	Waver numbers not to be used for fitting	
	allowed:	integer, string, any array
	Default:	[]
clipthresh	Clipping threshold in unit of sigma	
	allowed:	float
	Default:	3.0
clipniter	Maximum number of iteration of clipping	
	allowed:	integer
	Default:	0
plot	Plot the fit and the residual (currently unavailable)	
	allowed:	bool
	Default:	False
getresidual	If False, return best-fit value instead of residual	
	allowed:	bool
	Default:	True
showprogress	Show progress status for large data	
	allowed:	bool
	Default:	True
minnrow	Minimum number of spectra to show progress status	
	allowed:	integer
	Default:	1000
outlog	Output the coefficients of the best-fit function to logger	
	allowed:	bool
	Default:	False
blfile	Name of text file in which the best-fit parameter values to be written	
	allowed:	string
	Default:	''

Returns
scantable

Example

```
scan = sd.scantable('OrionS_rawACSmod_cal',average=False)
# return a scan baselined by a combination of sinusoidal curves having
# wave numbers in spectral window up to 10,
# also with 3-sigma clipping, iteration up to 4 times
bscan = scan.sinusoid_baseline(addwn='<=10',clipthresh=3.0,clipniter=4)
```

sd.scantable.smooth.html

sd.scantable.smooth - Function

5.1.2 Smooth the spectra

Description

Smooth the spectrum by the specified kernel (conserving flux).
The user should specify a type of smoothing kernel. Supported types are 'hanning' (default), 'gaussian', 'boxcar', 'rmedian', or 'poly'. The width of the kernel in pixel can be specified as width argument. For hanning this is ignored otherwise it defauls to 5 pixels. For 'gaussian' it is the Full Width Half Maximum. For 'boxcar' it is the full width. For 'rmedian' and 'poly' it is the half width. The order argument is an optional parameter for 'poly' kernel (default is 2), to specify the order of the polnomial. It is ignored by all other kernels.
You can verify and decide whether you apply the fit result or not, if plot argument is True. In that case, you have to answer 'y' or 'n' for each spectra so that setting True is not recommended for large dataset.
If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.

Arguments

Inputs		
kernel	The type of smoothing kernel	
	allowed:	string
	Default:	'hanning'
width	The width of the kernel in pixels	
	allowed:	float
	Default:	5.0
order	Order of the polynomial for 'poly' kernel	
	allowed:	integer
	Default:	2
plot	Plot the fit and the residual	
	allowed:	bool
	Default:	False
insitu	If False a new scantable is returned	
	allowed:	bool
	Default:	None

Returns
scantable

Example

sd.scantable.stats.html

sd.scantable.stats - Function

5.1.2 Compute specified statistics of the spectra

Description

Determine the specified statistic of the current beam/if/pol Takes a 'mask' as an optional parameter to specify which channels should be excluded.
Available statistics are 'min', 'max', 'min_abc', 'max_abc', 'sumsq', 'sum', 'mean', 'var', 'stddev', 'avdev', 'rms', 'median'.

Arguments

Inputs		
stat	Statistics to be calculated	
	allowed:	string
	Default:	'stddev'
mask	An optional mask specifying where the statistic should be determined	
	allowed:	bool array
	Default:	None
form	Format string to print statistic values	
	allowed:	string
	Default:	'3.3f'
row	Row number of spectrum to process	
	allowed:	integer
	Default:	None (all rows)

Returns

float array

Example

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

sd.scantable.stddev.html

sd.scantable.stddev - Function

5.1.2 Compute standard deviation of the spectra

Description

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

Arguments

Inputs		
mask	An optional mask specifying where the standard deviation should be determined	
	allowed:	bool array
	Default:	None

Returns
float array

Example

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

sd.scantable.summary.html

sd.scantable.summary - Function

5.1.2 Print a summary of the contents of the scantable

Description

Print a summary of the contents of this scantable. Optionally, output to the file specified.

Arguments

Inputs		
filename	The name of a file to write the output	
	allowed:	string
	Default:	None (no file output)

Returns
string

Example

sd.scantable.swap_linears.html

sd.scantable.swap_linears - Function

5.1.2 Swap the linear polarizations XX and YY

Description

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

Arguments

Returns

Example

sd.selector-Tool.html

5.1.3 sd.selector - Tool

Data selection tool for single-dish data

Description

The selector is a data selection object that affects a scantable object via set_selection function. Selection by the following attributes are available:

- scan number
- beam number
- cycle number
- IF number
- source name (allows regular expression)
- polarization number
- row number
- source type
- value of system temperature (minimum and maximum threshold)

Note that the selection by an integer is 0-based (scan, beam, cycle, IF, polarization, and row).
In addition, the selector provides interface for more flexible data selection using TaQL.
The selector also support sorting data. The selector sorts data based on values of columns specified by the user.
Summary of the current selection is stored in the object as a string. Thus, you can see that summary by using print (see example below).
The constructor takes some arguments to set selection criteria. So, the user can set selection either using constructor options and available setter functions.
Source types should be given as an integer or an enumerations that are properly defined for source type. That enumeration can be accessed via sd.srctype. The list of enumerations defined (following sd.srctype.) are as follows:

enum	int	description

pson	0	on-source scan of position switched data
psoff	1	off-source scan of position switched data
nod	2	nod scan
fson	3	on-source scan of frequency switched data
fsoff	4	reference scan of frequency switched data
sky	6	sky scan for calibration
hot	7	hot load scan for calibration
warm	8	warm load scan for calibration
cold	9	cold load scan for calibration
poncal	10	on-source calibration scan of position switched data
poffcal	11	off-source calibration scan of position switched data
nodcal	12	nod calibration scan
foncal	13	on-source calibration scan of frequency switched data
foffcal	14	reference calibration scan of frequency switched data
fslo	20	lower frequency throw of symmetric freuqnecy switching
flooff	21	off-source lower frequency throw of symmetric frequency switching
flosky	26	sky lower frequency throw of symmetric frequency switching
flohot	27	hot load lower frequency throw of symmetric frequency switching
flowarm	28	warm load lower frequency throw of symmetric frequency switching
flocold	29	cold load lower frequency throw of symmetric frequency switching
fshi	30	higher frequency throw of symmetric freuqnecy switching
fhioff	31	off-source higher frequency throw of symmetric frequency switching
fhisky	36	sky higher frequency throw of symmetric frequency switching
fhihot	37	hot load higher frequency throw of symmetric frequency switching
fhiwarm	38	warm load higher frequency throw of symmetric frequency switching
fhicold	39	cold load higher frequency throw of symmetric frequency switching
sig	90	signal scan
ref	91	reference scan
cal	92	calibration scan
notype	99	no type

Arguments

Inputs		
beams	Beam number selection	
	allowed:	integer, integer array
	Default:	None (no selection)
cycles	Cycle number selection	
	allowed:	integer, integer array
	Default:	None (no selection)
ifs	IF number selection	
	allowed:	integer, integer array
	Default:	None (no selection)
name	Source name selection. Allows regular expression.	
	allowed:	string
	Default:	None (no selection)
pols	Polarization number selection	
	allowed:	integer, integer array, string, string array
	Default:	None (no selection)
query	TaQL selection query	
	allowed:	string
	Default:	None (no selection)
scans	Scan number selection	
	allowed:	integer, integer array
	Default:	None (no selection)
types	Source type selection	
	allowed:	integer, integer array
	Default:	None (no selection)
rows	Row number selection	
	allowed:	integer, integer array
	Default:	None (no selection)

Example

```
# create a selector object
sel=sd.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()
# these are equivalent
sel2=sd.selector(ifs=[0,1])
sel2=sd.selector()
sel2.set_ifs([0,1])
# check current selection
```

```
print sel
IFNO: [0,1]
# apply selection on scantable
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
s.set_selection(sel)
```

Methods

get_beams	Return a list of current beam selection
get_cycles	Return a list of current cycle selection
get_ifs	Return a list of current IF selection
get_name	Return a selection string for source name (not yet implemented)
get_order	Return a list of columns used for sorting
get_pols	Return a list of current polarization selection
get_poltypes	Return a list of polarization types for current polarization selection
get_query	Return current selection string for TaQL selection
get_rows	Return a list of current row number selection
get_scans	Return a list of current scan number selection
get_types	Return a list of current source type selection
is_empty	Check if any selection has been set
reset	Unset all selections
set_beams	Set a sequence of beam numbers
set_cycles	Set a sequence of cycle numbers
set_ifs	Set a sequence of IF numbers
set_name	Set a selection based on a source name
set_order	Set column names that is used for sorting data
set_polarisations	Set a sequence of polarization numbers or strings
set_polarizations	Set a sequence of polarization numbers or strings
set_pols	Set a sequence of polarization numbers or strings
set_query	Set a selection string based on TaQL
set_rows	Set a sequence of row numbers
set_scans	Set a sequence of scan numbers
set_tsys	Set range of system temperature
set_types	Set a sequence of source types

sd.selector.get_beams.html

sd.selector.get_beams - Function

5.1.3 Return a list of current beam selection

Description

Return a list of current beam selection.

Arguments

Returns
integer array

Example

sd.selector.get_cycles.html

sd.selector.get_cycles - Function

5.1.3 Return a list of current cycle selection

Description

Return a list of current cycle selection.

Arguments

Returns
integer array

Example

sd.selector.get_ifs.html

sd.selector.get_ifs - Function

5.1.3 Return a list of current IF selection

Description

Return a list of current IF selection.

Arguments

Returns
integer array

Example

sd.selector.get_name.html

sd.selector.get_name - Function

5.1.3 Return a selection string for source name (not yet implemented)

Description

Return a selection strings for source name. Not yet implemented.

Arguments

Returns
None

Example

sd.selector.get_order.html

sd.selector.get_order - Function

5.1.3 Return a list of columns used for sorting

Description

Return a list of columns used for sorting.

Arguments

Returns
string list

Example

sd.selector.get_pols.html

sd.selector.get_pols - Function

5.1.3 Return a list of current polarization selection

Description

Return a list of current polarization selection. It returns an integer array even if you set polarization selection by string (e.g. 'XX', 'I').

Arguments

Returns
integer array

Example

sd.selector.get_poltypes.html

sd.selector.get_poltypes - Function

5.1.3 Return a list of polarization types for current polarization selection

Description

Return a list of polarization types for current polarization selection.
The returned array contains polarization types for each polarization selection strings. If you set polarization selection as integer, empty array will be returned since integer selection doesn't specify polarization type.

Arguments

Returns
string array

Example

sd.selector.get_query.html

sd.selector.get_query - Function

5.1.3 Return current selection string for TaQL selection

Description

Return current selection string for TaQL selection.

Arguments

Returns
string

Example

sd.selector.get_rows.html

sd.selector.get_rows - Function

5.1.3 Return a list of current row number selection

Description

Return a list of current row number selection.

Arguments

Returns
integer array

Example

sd.selector.get_scans.html

sd.selector.get_scans - Function

5.1.3 Return a list of current scan number selection

Description

Return a list of current scan number selection.

Arguments

Returns
integer array

Example

sd.selector.get_types.html

sd.selector.get_types - Function

5.1.3 Return a list of current source type selection

Description

Return a list of current source type selection.

Arguments

Returns
integer array

Example

sd.selector.is_empty.html

sd.selector.is_empty - Function

5.1.3 Check if any selection has been set

Description

Check if any selection has been set.
If any selection is set, the method returns False. Otherwise, it returns True.

Arguments

Returns
bool

Example

```
# create selector without any selection
sel=sd.selector()
# is_empty() returns True
sel.is_empty()
True
# set any selection
sel.set_ifs([0])
# then, is_empty() returns False
sel.is_empty()
False
```

sd.selector.reset.html

sd.selector.reset - Function

5.1.3 Unset all selections

Description

Unset all selections. Note that the method doesn't clear sorting order set by set_order. To unset sort order, you have to execute set_order([]).

Arguments

Returns

Example

```
# create selector without any selection
sel=sd.selector()
# is_empty() returns True
sel.is_empty()
True
# set any selection
sel.set_ifs([0])
# then, is_empty() returns False
sel.is_empty()
False
# reset() clear all selections
sel.reset()
# thus, is_empty() returns True
sel.is_empty()
True
```

sd.selector.set_beams.html

sd.selector.set_beams - Function

5.1.3 Set a sequence of beam numbers

Description

Set a sequence of beam numbers to the selection.

Arguments

Inputs		
beams	Beam number selection	
	allowed:	integer, integer array
	Default:	[] (no selection)

Returns

Example

sd.selector.set_cycles.html

sd.selector.set_cycles - Function

5.1.3 Set a sequence of cycle numbers

Description

Set a sequence of cycle numbers to the selection.

Arguments

Inputs		
cycles	Cycle number selection	
	allowed:	integer, integer array
	Default:	[] (no selection)

Returns

Example

sd.selector.set_ifs.html

sd.selector.set_ifs - Function

5.1.3 Set a sequence of IF numbers

Description

Set a sequence of IF numbers to the selection.

Arguments

Inputs		
ifs	IF number selection	
	allowed:	integer, integer array
	Default:	[] (no selection)

Returns

Example

sd.selector.set_name.html

sd.selector.set_name - Function

5.1.3 Set a selection based on a source name

Description

Set a selection based on a source name. This can be a unix pattern , e.g. "*_R".
Note that the method overwrites TaQL selection string set by set_query.

Arguments

Inputs		
name	A string containing a source name or pattern	
	allowed:	string
	Default:	

Returns

Example

```
sel=sd.selector()
# select all reference scans which start with "Orion"
sel.set_name("Orion*_R")
```

sd.selector.set_order.html

sd.selector.set_order - Function

5.1.3 Set column names that is used for sorting data

Description

Set the order the scantable should be sorted by. The user specify a list of column names that is used for sorting data (e.g. 'IFNO', 'SCANNO'). The data are sorted according to the value of these columns.

Arguments

Inputs		
order	The list of column names to sort by in order	
	allowed:	string array
	Default:	

Returns

Example

sd.selector.set_polarisations.html

sd.selector.set_polarisations - Function

5.1.3 Set a sequence of polarization numbers or strings

Description

Set the polarisations to be selected in the scantable. It allows to set polarization selection via integer or string (e.g. "XX", "Q"). Integer must be within the range of 0 to 3 since the number outside this range will result no selected data exception. If the selection is specified by an integer, the selection will refer polarization type of the original scantable. On the other hand, the selection by string contains both type and component so that the selection will refer its type, not the original scantable.

Arguments

Inputs		
pols	A list of integers of 0-3, or strings, e.g ["I","Q"].	
	allowed:	integer, string, integer array, string array
	Default:	[] (no selection)

Returns

Example

```
sel = sd.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()
```

sd.selector.set_polarizations.html

sd.selector.set_polarizations - Function

5.1.3 Set a sequence of polarization numbers or strings

Description

See set_polarisations.

Arguments

Returns

Example

sd.selector.set_pols.html

sd.selector.set_pols - Function

5.1.3 Set a sequence of polarization numbers or strings

Description

See set_polarisations.

Arguments

Returns

Example

sd.selector.set_query.html

sd.selector.set_query - Function

5.1.3 Set a selection string based on TaQL

Description

Select by Column query. Power users only!
The mehtod is used to set TaQL selection string directly. The user must be familiar with TaQL and data structure of scantable.

Arguments

Inputs		
query	TaQL selection string	
	allowed:	string
	Default:	

Returns

Example

```
sel=sd.selector()
# select all off scans with integration times over 60 seconds.
sel.set_query("SRCTYPE == PSOFF AND INTERVAL > 60.0")
```

sd.selector.set_rows.html

sd.selector.set_rows - Function

5.1.3 Set a sequence of row numbers

Description

Set a sequence of row numbers (0-based). Power users Only!
NOTICE row numbers can be changed easily by sorting, prior selection, etc.

Arguments

Inputs		
rows	A list of integers	
	allowed:	integer, integer array
	Default:	[] (no selection)

Returns

Example

sd.selector.set_scans.html

sd.selector.set_scans - Function

5.1.3 Set a sequence of scan numbers

Description

Set a sequence of Scan numbers to the selection.

Arguments

Inputs		
scans	A list of integers	
	allowed:	integer, integer array
	Default:	[] (no selection)

Returns

Example

sd.selector.set_tsys.html

sd.selector.set_tsys - Function

5.1.3 Set range of system temperature

Description

Select by Tsys range. The method sets an upper and a lower limit of the Tsys value.

Arguments

Inputs		
tsysmin	The lower threshold	
	allowed:	float
	Default:	0.0
tsysmax	The upper threshold	
	allowed:	float
	Default:	None (no upper limit)

Returns

Example

```
sel=sd.selector()
# select all spectra with Tsys <= 500.0
sel.set_tsys(tsysmax=500.0)
```

sd.selector.set_types.html

sd.selector.set_types - Function

5.1.3 Set a sequence of source types

Description

Set a sequence of source types to the selection. The types argument can contain integer and/or sd.srctype enum.

Arguments

Inputs		
types	A list of integers	
	allowed:	integer, integer array
	Default:	[] (no selection)

Returns

Example

```
sel=sd.selector()
# select only on-source scans of position switched observation
sel.set_types(0)
# or use enum
sel.set_types(sd.srctype.pson)
```

sd.fitter-Tool.html

5.1.4 sd.fitter - Tool

General fitting tool

Description

The fitter is an object to fit data. This contains both baseline fitting and spectral line fitting. For baseline fitting, simple polynomial fitting with arbitrary order is available. For spectral line fitting, Gaussian and Lorentzian fitting are supported. It allows fitting of multiple lines, but it doesn't support hyperfine fitting, i.e. multiple line fitting under some constraints.
The constructor doesn't take any arguments. It just creates a fitter object that no state is set.

Example

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

Methods

auto_fit	Return a scan where the function is applied to all rows
commit	Return a new scan where the fits have been subtracted
fit	Execute actual fitting process
get_area	Return the area under the fitted gaussian/lorentzian component
get_chi2	Return χ^2 value
get_errors	Return the errors in parameters
get_estimate	Return the parameter estimates
get_fit	Return the fitted ordinate values
get_parameters	Return the fit parameters
get_residual	Return the residual of the fit
plot	Plot fit
set_data	Set the abcissa and ordinate for the fit
set_function	Set the function to be fitted
set_gauss_parameters	Set the parameters of Gaussian component
set_lorentz_parameters	Set the parameters of Lorentzian component
set_parameters	Set the parameters to be fitted
set_scan	Set the data as a scantable

set_sinusoid_parameters	Set the parameters of Sinusoidal component
store_fit	Save fit parameters

sd.fitter.auto_fit.html

sd.fitter.auto_fit - Function

5.1.4 Return a scan where the function is applied to all rows

Description

This method executes fitting for all rows in the current selection of the data. It returns a scan where the function is applied to all rows for all Beams/IFs/Pols. If any data is not set and/or fitting function is not specified, the function throws an exception.

Arguments

Inputs		
insitu	Apply operation on the input scantable or not	
	allowed:	bool
	Default:	None (use default setting defined in sd.rcParams)
plot	Plot and verify result or not	
	allowed:	bool
	Default:	False

Returns
scantable

Example

```
f=sd.fitter()
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
f.set_scan(s)
f.set_function(poly=3)
s_bs=f.auto_fit()
```

sd.fitter.commit.html

sd.fitter.commit - Function

5.1.4 Return a new scan where the fits have been subtracted

Description

Return a new scan where the fits have been subtracted. You must fit the data before you execute this method. Otherwise, an exception will be thrown.
If the data is set as a set of abcissa and ordinate values using set_data, it will not work. If you want to get data by this method, you must use set_scan that set the data as a scantable.

Arguments

Returns
scantable

Example

```
f=sd.fitter()
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
# you must set data as a scantable
f.set_scan(s)
# set polynomial function of order 3 for baseline subtraction
f.set_function(poly=3)
# do fit
f.fit()
# get baseline subtracted scans
s_bs=f.commit()
```

sd.fitter.fit.html

sd.fitter.fit - Function

5.1.4 Execute actual fitting process

Description

Execute the actual fitting process. All the state (data, selection, and function) has to be set before executing.
The method executes a fitting process for only one row that are specified by a row argument. By default, the first row will be fitted.
The estimate argument determines if the method computes an initial parameter set automatically. This can be used to compute estimates even if fit was called before.

Arguments

Inputs		
row	Specify the row in the scantable	
	allowed:	integer
	Default:	0
estimate	Auto-compute an initial parameter set	
	allowed:	bool
	Default:	False

Returns

Example

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

sd.fitter.get_area.html

sd.fitter.get_area - Function

5.1.4 Return the area under the fitted gaussian/lorentzian component

Description

Return the area under the fitted gaussian/lorentzian component. The component argument specifies which component you want to get area. By default, the method will return a sum of all gaussian/lorentzian components. Note that this will only work for gaussian/lorentzian fits. Otherwise, None will be returned.

Arguments

Inputs		
component	The gaussian/lorentzian component selection	
	allowed:	integer
	Default:	None (sum of all components)

Returns
float

Example

sd.fitter.get_chi2.html

sd.fitter.get_chi2 - Function

5.1.4 Return χ^2 value

Description

Return χ^2 value of the fit. You must execute fitting process before getting χ^2.

Arguments

Returns
float

Example

```
s = scantable('myscan.asap')
s.set_cursor(thepol=1)        # select second pol (nb. 0-based)
f = fitter()
f.set_scan(s)
f.set_function(poly=0)
f.fit(row=0)                  # fit first row
ch2=f.get_chi2()
```

sd.fitter.get_errors.html

sd.fitter.get_errors - Function

5.1.4 Return the errors in parameters

Description

Return the errors in the parameters. You must execute fitting process before getting errors.

Arguments

Inputs		
component	get the errors for the specified component only	
	allowed:	integer
	Default:	None (all components)

Returns
float array

Example

sd.fitter.get_estimate.html

sd.fitter.get_estimate - Function

5.1.4 Return the parameter estimates

Description

Return the parameter estimates for non-linear functions. It works only if fit is executed with estimate=True.

Arguments

Returns
float array

Example

sd.fitter.get_fit.html

sd.fitter.get_fit - Function

5.1.4 Return the fitted ordinate values

Description

Return the fitted ordinate values. For spectral line fitting, it will represent model of the observed line. You must execute fitting process before getting fitted result.

Arguments

Returns
float array

Example

sd.fitter.get_parameters.html

sd.fitter.get_parameters - Function

5.1.4 Return the fit parameters

Description

Return the fit parameters as a dictionary. The returned value contains the following attributes:

- errors
  errors for each parameter
- fixed
  list of fixed parameters
- formatted
  formatted string that shows a fitting result
- params
  list of resulting parameter values

The component argument specifies which component the user want to get fit parameters. It is effective only for gaussian/lorentzian fitting.
The error argument controls contents of formatted string. If error is True, the string contains parameter values and their errors.

Arguments

Inputs		
component	get the parameter values for the specified component only	
	allowed:	integer
	Default:	None (all components)
errors	If True, include errors of fit parameters in formatted string	
	allowed:	bool
	Default:	False

Returns
dictionary

Example

sd.fitter.get_residual.html

sd.fitter.get_residual - Function

5.1.4 Return the residual of the fit

Description

Return the residual of the fit. For baseline fitting, it will represent a baseline-subtracted spectrum. You must execute fitting process before getting fitted result.

Arguments

Returns
float array

Example

sd.fitter.plot.html

sd.fitter.plot - Function

5.1.4 Plot fit

Description

Plot the last fit.
There are three arguments that control plot. If residual argument is True, the plot contains residual in addition to original data and fit. If plotparms argument is True, the parameter values will be written on the plot explicitly. The components argument specifies a list of components to plot.
components=-1 means total fit (sum of all components) will be plotted.

Arguments

Inputs	
residual	An optional parameter indicating if the residual should be plotted
	allowed: bool
	Default: False
components	a list of components to plot, e.g [0,1]
	allowed: integer, integer array
	Default: None (plot total fit)
plotparms	Inidicates if the parameter values should be present on the plot
	allowed: bool
	Default: False

Returns

Example

sd.fitter.set_data.html

sd.fitter.set_data - Function

5.1.4 Set the abcissa and ordinate for the fit

Description

Set the absissa and ordinate for the fit. Also set the mask indicationg valid points. This can be used for data vectors retrieved from a scantable.
For scantable fitting use set_scan.

Arguments

Inputs		
xdat	The abcissa values	
	allowed:	float array
	Default:	
ydat	The ordinate values	
	allowed:	float array
	Default:	
mask	An optional mask	
	allowed:	bool array
	Default:	

Returns

Example

sd.fitter.set_function.html

sd.fitter.set_function - Function

5.1.4 Set the function to be fitted

Description

Set the function to be fit.
The argument to be set determines what the fitter will do. If you want to do polynomial fitting, you have to set poly or lpoly arguments. The value of poly or lpoly are interpreted as an order of the polynomial function to be used for fitting. The poly is for non-linear least squares fitting, while lpoly is for linear one. If you want to do line fitting, you have to set either gauss or lorentz arguments. In that case, values of gauss or lorentz arguments are interpreted as a number of gaussian/lorentzian components. Apparently, gauss=0 or lorentz=0 causes an exception.
Note that all the above arguments are exclusive each other.

Arguments

Inputs		
poly	Use a polynomial of the order given with nonlinear least squares fit	
	allowed:	integer
	Default:	
lpoly	Use a polynomial of the order given with linear least squares fit	
	allowed:	integer
	Default:	
gauss	Fit the number of gaussian specified	
	allowed:	integer
	Default:	
lorentz	Fit the number of lorentzian specified	
	allowed:	integer
	Default:	

Returns

Example

```
f=sd.fitter()
# will fit a 3rd order polynomial via nonlinear method
f.set_function(poly=3)
# will fit a 3rd order polynomial via linear method
f.set_function(lpoly=3)
# will fit two gaussians
f.set_function(gauss=2)
# will fit two lorentzians
f.set_function(lorentz=2)
```

sd.fitter.set_gauss_parameters.html

sd.fitter.set_gauss_parameters - Function

5.1.4 Set the parameters of Gaussian component

Description

Set the Parameters of a 'Gaussian' component, set with set_function.
Three arguments, peak, centre, and fwhm, are mandatory to be set. They specifies an initial model of Gaussian component.
The peakfixed, centrefixed, and fwhmfixed are optional parameters to indicate if the paramters should be held fixed during the fitting process. The default is to keep all parameters flexible. If you want to fix one of those parameters, you can do it by setting corresponding arguments (peakfixed, centrefixed, and fwhmfixed) to 1.
The component argument is only effective for multi-component Gaussian fitting. It specifies the number of the component to set the specified parameters.

Arguments

Inputs	
peak	The gaussian parameters (peak intensity) allowed: float Default:
centre	The gaussian parameters (line center) allowed: float Default:
fwhm	The gaussian parameters (FWHM) allowed: float Default:
peakfixed	Optional parameters to indicate if peak should be held fixed during the fitting process allowed: integer Default: 0 (flexible)
centrefixed	Optional parameters to indicate if centre should be held fixed during the fitting process allowed: integer Default: 0 (flexible)
fwhmfixed	Optional parameters to indicate if fwhm should be held fixed during the fitting process allowed: integer Default: 0 (flexible)
component	The number of the component allowed: integer Default: 0

Returns

Example

```
f=sd.fitter()
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
f.set_scan(s)
# set fit function as gaussian
f.set_function(gauss=1)
# set gaussian parameter with centre fixed
f.set_gauss_parameters(peak=0.5,centre=4100,fwhm=200,centrefixed=1)
f.fit()
```

sd.fitter.set_lorentz_parameters.html

sd.fitter.set_lorentz_parameters - Function

5.1.4 Set the parameters of Lorentzian component

Description

Set the Parameters of a 'Lorentzian' component, set with set_function.
Three arguments, peak, centre, and fwhm, are mandatory to be set. They specifies an initial model of Lorentzian component.
The peakfixed, centrefixed, and fwhmfixed are optional parameters to indicate if the paramters should be held fixed during the fitting process. The default is to keep all parameters flexible. If you want to fix one of those parameters, you can do it by setting corresponding arguments (peakfixed, centrefixed, and fwhmfixed) to 1.
The component argument is only effective for multi-component Lorentzian fitting. It specifies the number of the component to set the specified parameters.

Arguments

Inputs		
peak	The lorentzian parameters (peak intensity)	
	allowed:	float
	Default:	
centre	The lorentzian parameters (line center)	
	allowed:	float
	Default:	
fwhm	The lorentzian parameters (FWHM)	
	allowed:	float
	Default:	
peakfixed	Optional parameters to indicate if peak should be held fixed during the fitting process	
	allowed:	integer
	Default:	0 (flexible)
centrefixed	Optional parameters to indicate if centre should be held fixed during the fitting process	
	allowed:	integer
	Default:	0 (flexible)
fwhmfixed	Optional parameters to indicate if fwhm should be held fixed during the fitting process	
	allowed:	integer
	Default:	0 (flexible)
component	The number of the component	
	allowed:	integer
	Default:	0

Returns

Example

```
f=sd.fitter()
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
f.set_scan(s)
# set fit function as lorentzian
f.set_function(lorentz=1)
# set lorentzian parameter with centre fixed
f.set_lorentz_parameters(peak=0.5,centre=4100,fwhm=200,centrefixed=1)
f.fit()
```

sd.fitter.set_parameters.html

sd.fitter.set_parameters - Function

5.1.4 Set the parameters to be fitted

Description

Set the parameters to be fitted.
It takes an argument that indicates a list of parameter values and preference if those parameters are fixed or flexible during the fit. These lists should be given as a dictionary. The params and fixed in the following argument list doesn't indicate argument itself here. Instead, they indicate keywords in the above dictionary.
This method is normally called from set_gauss_parameters, set_lorentz_parameters, and set_sinusoid_parameters so that you may not need to call this method directly.

Arguments

Inputs		
params	A vector of parameters (peak, centre, fwhm)	
	allowed:	float array
	Default:	
fixed	A vector of which parameters are to be held fixed	
	allowed:	float array
	Default:	None (all parameters are flexible)
component	In case of multiple gaussians/lorentzians, the index of the component	
	allowed:	integer
	Default:	0

Returns

Example

sd.fitter.set_sinusoid_parameters.html

sd.fitter.set_sinusoid_parameters - Function

5.1.4 Set the parameters of Sinusoidal component

Description

Set the Parameters of a 'Sinusoidal' component, set with set_function.
Three arguments, ampl, period, and x0, are mandatory to be set. They specifies an initial model of Sinusoidal component.
The amplfixed, periodfixed, and x0fixed are optional parameters to indicate if the paramters should be held fixed during the fitting process. The default is to keep all parameters flexible. If you want to fix one of those parameters, you can do it by setting corresponding arguments (amplfixed, periodfixed, and x0fixed) to 1.
The component argument is only effective for multi-component Sinusoidal fitting. It specifies the number of the component to set the specified parameters.

Arguments

Inputs		
ampl	The sinusoidal parameters (amplitude)	
	allowed:	float
	Default:	
period	The sinusoidal parameters (period)	
	allowed:	float
	Default:	
x0	The sinusoidal parameters (phase offset)	
	allowed:	float
	Default:	
amplfixed	Optional parameters to indicate if ampl should be held fixed during the fitting process	
	allowed:	integer
	Default:	0 (flexible)
periodfixed	Optional parameters to indicate if period should be held fixed during the fitting process	
	allowed:	integer
	Default:	0 (flexible)
x0fixed	Optional parameters to indicate if x0 should be held fixed during the fitting process	
	allowed:	integer
	Default:	0 (flexible)
component		
	allowed:	integer
	Default:	0

Returns

Example

sd.fitter.set_scan.html

sd.fitter.set_scan - Function

5.1.4 Set the data as a scantable

Description

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

Arguments

Inputs		
thescan	A scantable	
	allowed:	scantable
	Default:	
mask	A retrieved from the scantable	
	allowed:	bool array
	Default:	

Returns

Example

sd.fitter.store_fit.html

sd.fitter.store_fit - Function

5.1.4 Save fit parameters

Description

Save the fit parameters.
If filename is specified, the method saves the result in an ASCII file.
Otherwise, the result will be stored in the scantable. More specifically, the result will be stored in FIT subtable in the scantable.
It works both for spectral line fitting (gaussian or lorentzian fitting) and polynomial fitting although it is not sure if the latter is useful or not.

Arguments

Inputs		
filename	If specified, save as an ASCII table	
	allowed:	string
	Default:	None (store result in the scantable)

Returns

Example

```
f=sd.fitter()
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
f.set_scan(s)
# set fit function as gaussian
f.set_function(gauss=1)
# set gaussian parameter with centre fixed
f.set_gauss_parameters(peak=0.5,centre=4100,fwhm=200,centrefixed=1)
f.fit()
# store fit in the scantable
f.store_fit()
# store fit in an ASCII file
f.store_fit(filename='fitresult.txt')
```

sd.linecatalog-Tool.html

5.1.5 sd.linecatalog - Tool

Line catalog

Description

The linecatalog is a wrapper for line catalog data base. These can be either ASCII tables or the tables saved from this class. The table consists of the following items:

- row index in the table
- name of the species
- line frequency
- frequency error
- line intensity

The user can filter lines by name, frequency range, or intensity range. It is possible to overlay line catalog on the spectral plot using plotter.plot_lines.
For ASCII type input table, Comments can be present through lines starting with '#'.
The constructor takes string that specifies a name of the catalog.

Known Issues
The name of species canno't contain spaces. If it does, it has to be wrapped in double-quotes.

Arguments

Inputs		
name	Name of the catalog	
	allowed:	string
	Default:	

Example

```
l = sd.linecatalog('jpl_asap.tbl')
# set name restriction
l.set_name('NH3')
# print summary
l.summary()
```

Methods

get_frequency	Get frequency in a specified row
get_name	Get name of specie in a specified row
get_row	Get the values in a specified row
nrow	Get number of rows in the table
reset	Unset all filtering to the table
save	Save the subset of the table to disk
set_frequency_limits	Set frequency limit on the table
set_name	Set a name restriction on the table
set_strength_limits	Set line strength limit on the table
summary	Print the contents of the table

sd.linecatalog.get_frequency.html

sd.linecatalog.get_frequency - Function

5.1.5 Get frequency in a specified row

Description

The method returns a frequency value in the catalog for given row number.

Arguments

Inputs		
	Row number	
	allowed:	integer
	Default:	

Returns
float

Example

sd.linecatalog.get_name.html

sd.linecatalog.get_name - Function

5.1.5 Get name of specie in a specified row

Description

The method returns a name of species in the catalog for given row number.

Arguments

Inputs		
	Row number	
	allowed:	integer
	Default:	

Returns
string

Example

sd.linecatalog.get_row.html

sd.linecatalog.get_row - Function

5.1.5 Get the values in a specified row

Description

The method returns the values in a specified row of the catalog. Returned value is a dictionary that contains a name of the species and its frequency for that row.

Arguments

Inputs		
row	The row to retrieve	
	allowed:	integer
	Default:	0

Returns
dictionary

Example

sd.linecatalog.nrow.html

sd.linecatalog.nrow - Function

5.1.5 Get number of rows in the table

Description

The method returns a number of rows of the catalog.

Arguments

Returns
integer

Example

sd.linecatalog.reset.html

sd.linecatalog.reset - Function

5.1.5 Unset all filtering to the table

Description

This method resets the table to its initial state, i.e. undo all calls of filtering methods on the catalog.

Arguments

Returns

Example

sd.linecatalog.save.html

sd.linecatalog.save - Function

5.1.5 Save the subset of the table to disk

Description

Save the subset of the table to disk. This uses an internal data format and can be read in again. If no filtering is done on the catalog, it just copies the original catalog.

Arguments

Inputs		
name	The name of the output catalog	
	allowed:	string
	Default:	
overwrite	Overwrite existing table if True	
	allowed:	bool
	Default:	False

Returns

Example

sd.linecatalog.set_frequency_limits.html

sd.linecatalog.set_frequency_limits - Function

5.1.5 Set frequency limit on the table

Description

Set frequency limits on the table.
Note that the underlying table contains frequency values in MHz

Arguments

Inputs		
fmin	The lower bound	
	allowed:	float
	Default:	1.0
fmax	The upper bound	
	allowed:	float
	Default:	120.0
unit	The frequency unit	
	allowed:	string ('GHz' or 'MHz')
	Default:	'GHz'

Returns

Example

sd.linecatalog.set_name.html

sd.linecatalog.set_name - Function

5.1.5 Set a name restriction on the table

Description

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

Arguments

Inputs		
name	The name pattern/regex	
	allowed:	string
	Default:	
mode	The matching mode, i.e. 'pattern' or 'regex'	
	allowed:	string
	Default:	'pattern'

Returns

Example

sd.linecatalog.set_strength_limits.html

sd.linecatalog.set_strength_limits - Function

5.1.5 Set line strength limit on the table

Description

Set line strength limits on the table (arbitrary units).

Arguments

Inputs		
smin	The lower bound	
	allowed:	float
	Default:	
smax	The upper bound	
	allowed:	float
	Default:	

Returns

Example

sd.linecatalog.summary.html

sd.linecatalog.summary - Function

5.1.5 Print the contents of the table

Description

Print the contents of the table.

Arguments

Returns

Example

```
l=sd.linecatalog('jpl_asap.tbl')
l.set_name('NH3')
l.summary()
# the output should look like the following
--------------------------------------------------------------------------------
Line Catalog summary
--------------------------------------------------------------------------------

      0  NH3                 10293.659   -7.9334
      1  NH3                 10426.949   -7.6822
      2  NH3                 10536.183   -7.3906
      3  NH3                 10836.127   -7.8889
      4  NH3                 11132.722   -7.6055
      5  NH3                 11673.171   -7.6559
      6  NH3                 11947.244   -7.2809
      7  NH3                 12251.33    -6.9818
      8  NH3                 12336.462   -7.4568
      9  NH3                 12923.04    -6.9522
     10  NH3                 12951.048   -7.4088
     11  NH3                 13296.342   -6.7014
     12  NH3                 13297.266   -7.316
     13  NH3                 13612.146   -7.5443
     14  NH3                 13700.883   -6.6744
```

```
15  NH3                 13719.23    -6.9543
16  NH3                 13974.605   -6.897
17  NH3                 14224.647   -6.3276
18  NH3                 14376.817   -6.1817
```

sd.linefinder-Tool.html

5.1.6 sd.linefinder - Tool

Line finder tool for single-dish spectrum

Description

The linefinder performs automated spectral line search. 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.
There are six parameters for the algorithm. These can be set by set_options method of this object.

- threshold
  A single channel S/N ratio above which the channel is considered to be a detection. Default is sqrt(3), which together with min_nchan=3 gives a 3-sigma criterion

- min_nchan
  A minimal number of consequtive channels, which should satisfy a threshold criterion to be a detection. Default is 3.

- avg_limit
  A number of consequtive channels not greater than this parameter can be averaged to search for broad lines. Default is 8.

- box_size
  A running mean/median box size specified as a fraction of the total spectrum length. Default is 1/5

- noise_box
  Area of the spectrum used to estimate noise stats. Both string values and numbers are allowed Allowed string values are 'all' that use all the spectrum (default), and 'box' means noise box is the same as running mean/median box. Numeric values are defined as a fraction from the spectrum size. Values should be positive. (noise_box == box_size has the same effect as noise_box = 'box')

- noise_stat
  Statistics used to estimate noise. Allowed values are 'mean80' that use the 80% of the lowest deviations in the noise box (default) and 'median' means median of deviations in the noise box.

The constructor doesn't take any arguments. It creates linefinder object without any settings for line finding.

Example

```
fl=sd.linefinder()
sc=sd.scantable('sddata.asap',average=False)
# set data
fl.set_scan(sc)
# set linefinder options
fl.set_options(threshold=3)
# search lines
nlines=fl.find_lines(edge=(50,0))
# get range of lines found
if nlines!=0:
   print "Found ",nlines," spectral lines"
   print fl.get_ranges(False)
else:
   print "No lines found!"
# baseline subtraction using masks provided by linefinder
sc2=sc.poly_baseline(fl.get_mask(),7)
```

Methods

find_lines	Search for spectral lines in the scan
get_mask	Get the mask to mask all lines that have been found
get_ranges	Get ranges for all spectral lines found
set_data	Set the data as an array
set_options	Set the parameters of the line finding algorithm
set_scan	Set the data as a scantable

sd.linefinder.find_lines.html

sd.linefinder.find_lines - Function

5.1.6 Search for spectral lines in the scan

Description

Search for spectral lines in the scan assigned in set_scan. A number of lines found will be returned.
The method allows to set optional masks for the search. The mask parameter is a bool array that sets channel-by-channel masking. On the other hand, the edge parameter set a number of channels to drop at the edge of the spectrum. The number can be set for each side as an integer array with length of two. If only one value is specified, the same number will be dropped from both sides of the spectrum.

Arguments

Inputs		
nRow	A row number in the scantable to work with	
	allowed:	integer
	Default:	0
mask	An optional mask	
	allowed:	bool array
	Default:	[] (no mask)
edge	An optional number of channels to drop at the edge of the spectrum	
	allowed:	integer, integer array
	Default:	(0,0) (keep all channels)

Returns
integer

Example

sd.linefinder.get_mask.html

sd.linefinder.get_mask - Function

5.1.6 Get the mask to mask all lines that have been found

Description

By default, the method returns the mask to mask out all lines that have been found. If the invert option is True, inverted mask that only channels belong to lines are unmasked, will be returned.
Note that all channels originally masked by the input mask or dropped out by the edge parameter will still be excluded regardless on the invert option.

Arguments

Inputs		
invert	If True, only channels belong to lines will be unmasked	
	allowed:	bool
	Default:	False

Returns
bool array

Example

sd.linefinder.get_ranges.html

sd.linefinder.get_ranges - Function

5.1.6 Get ranges for all spectral lines found

Description

The method returns ranges (start and end channels or velocities) for all spectral lines found.
By default, the unit of returned values will be used the same unit as set for the scan that is set by this object using set_scan. If defunits option is set to False, the range will always be expressed in channels.

Arguments

Inputs		
defunits	If False, the returned range will be expressed in channels	
	allowed:	bool
	Default:	True

Returns
bool array

Example

sd.linefinder.set_data.html

sd.linefinder.set_data - Function

5.1.6 Set the data as an array

Description

Set the 'data' (spectrum) to work with Parameters: a method to allow linefinder work without setting scantable for the purpose of using linefinder inside some method in scantable class. (Dec 22, 2010 by W.Kawasaki)

Arguments

Inputs		
spectrum	The data to be set	
	allowed:	float array
	Default:	

Returns

Example

sd.linefinder.set_options.html

sd.linefinder.set_options - Function

5.1.6 Set the parameters of the line finding algorithm

Description

This method is used to set the parameters of the line finding algorithm. There are six parameters that can be set by this method. See description of this object for details about parameters.
Note that, 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.

Arguments

Inputs	
threshold	A single channel S/N ratio above which the channel is considered to be a detection. allowed: float Default: sqrt(3)
min_nchan	A minimal number of consequtive channels, which should satisfy a threshold criterion to be a detection. allowed: integer Default: 3
avg_limit	A number of consequtive channels not greater than this parameter can be averaged to search for broad lines. allowed: integer Default: 8
box_size	A running mean/median box size specified as a fraction of the total spectrum length. allowed: float Default: 0.2
noise_box	Area of the spectrum used to estimate noise stats allowed: float, string ('all' or 'box') Default: 'all'
noise_stat	Statistics used to estimate noise allowed: string ('mean80' or 'median') Default: 'mean80'

Returns

Example

sd.linefinder.set_scan.html

sd.linefinder.set_scan - Function

5.1.6 Set the data as a scantable

Description

The method sets the data to work with. The data must be given as a scantable.

Arguments

Inputs		
scan	The data to be set	
	allowed:	scantable
	Default:	

Returns

Example

sd.simplelinefinder-Tool.html

5.1.7 sd.simplelinefinder - Tool

Simplified line finder tool for single-dish spectrum

Description

A simplified class to search for spectral features. The algorithm assumes that the bandpass is taken out perfectly and no spectral channels are flagged (except some edge channels). It works with a list or tuple rather than a scantable and returns the channel pairs. There is an optional feature to attempt to split the detected lines into components, although it should be used with caution. This class is largely intended to be used with scripts. The fully featured version of the algorithm working with scantables is called linefinder.
Methods

channelRange	Convert supplied velocity range into the channel range
find_lines	Search for spectral lines in the spectrum
invertChannelSelection	Invert channel range selection
median	Return a median of the last spectrum passed to find_lines
rms	Return rms calculated during last find_lines call
writeLog	Write user defined string into log file

sd.simplelinefinder.channelRange.html

sd.simplelinefinder.channelRange - Function

5.1.7 Convert supplied velocity range into the channel range

Description

A helper method which works on a tuple with abcissa/flux vectors (i.e. as returned by uvSpectrum). It allows to convert supplied velocity range into the channel range.
The argument spc specifies abcissa/flux vectors as tuple. First and second elements of the tuple should be abcissa value and spectrum itself, respectively.
The argument vel_range must be 2-element tuple that indicates start and stop velocity of range.
Note, if supplied range is completely outside the spectrum, an empty tuple will be returned.

Arguments

Inputs	tuple with the abcissa and spectrum	
spc		
	allowed:	tuple
	Default:	
vel_range	a 2-element tuple with start and stop velocity of the range	
	allowed:	tuple
	Default:	

Returns
a 2-element tuple with channels

Example

sd.simplelinefinder.find_lines.html

sd.simplelinefinder.find_lines - Function

5.1.7 Search for spectral lines in the spectrum

Description

A simple spectral line detection routine, which assumes that bandpass has been taken out perfectly and no channels are flagged within the spectrum. A detection is reported if consequtive minchan number of channels is consistently above or below the median value. The threshold is given in terms of the rms calculated using 80% of the lowest data points by the absolute value (with respect to median).
This method returns a list of tuples each containing start and stop 0-based channel number of every detected line. Empty list if nothing has been detected.
Note. The median and rms about this median is stored inside this class and can be obtained with rms and median methods.

Arguments

Inputs		
spc	a list or tuple with the spectrum, no default	
	allowed:	tuple
	Default:	
edge	detection threshold	
	allowed:	intger
	Default:	0
minchan	minimum number of consequitive channels exceeding threshold to claim the detection	
	allowed:	integer
	Default:	3
tailsearch	if True (default), the algorithm attempts to widen each line until its flux crosses the median. It merges lines if necessary	
	allowed:	bool
	Default:	True
splitFeatures	if True, the algorithm attempts to split each detected spectral feature into a number of spectral lines (just one local extremum)	
	allowed:	bool
	Default:	False

Returns
list of tuples that indicates ranges of detected lines

Example

sd.simplelinefinder.invertChannelSelection.html

sd.simplelinefinder.invertChannelSelection - Function

5.1.7 Invert channel range selection

Description

This method converts a tuple with channel ranges to a tuple which covers all channels not selected by the original tuple (optionally edge channels can be discarded).
Note, at this stage channel ranges are assumed to be sorted and without overlap.

Arguments

Inputs		
nchan	number of channels in the spectrum	
	allowed:	integer
	Default:	
chans	list of start and stop channels for all selected ranges	
	allowed:	tuple
	Default:	
edge	how many channels to reject from the edge of spectrum	
	allowed:	integer or tuple
	Default:	(0,0)

Returns
tuple with inverted channel selection

Example

sd.simplelinefinder.median.html

sd.simplelinefinder.median - Function

5.1.7 Return a median of the last spectrum passed to find_lines

Description

Return the median of the last spectrum passed to find_lines. Note, this method throws an exception if find_lines has never been called.

Arguments

Returns
float

Example

sd.simplelinefinder.rms.html

sd.simplelinefinder.rms - Function

5.1.7 Return rms calculated during last find_lines call

Description

Return rms scatter of the spectral points (with respect to the median)
calculated during last find_lines call. Note, this method throws an exception if
find_lines has never been called.

Arguments

Returns
float

Example

sd.simplelinefinder.writeLog.html

sd.simplelinefinder.writeLog - Function

5.1.7 Write user defined string into log file

Description

Write user defined string into log file.

Arguments

Inputs		
str	log message	
	allowed:	string
	Default:	

Returns

Example

sd.plotter-Tool.html

5.1.8 sd.plotter - Tool

Single-dish specific plotter tool

Description

This is a plotter object that is properly designed for single-dish tool. It supports stacking, multi-panel plotting, and multi-page plotting of spectral plot (channel/frequency/velocity versus spectral data). For total power (single-channel) data, it provides special function to plot total power data versus time. It also supports to plot time variation of azimuth and elevation and to plot pointing directions on the sky.

Known Issues
The multi-page plotting doesn't support to go back previous page at the moment, it only allows to go forward.
Methods

annotate	Annotate text at specified location
arrow	Draw arrow on specified axis
axhline	Draw a horizontal line
axhspan	Draw a horizontal rectangle
axvline	Draw a vertical line
axvspan	Draw a vertical rectangle
casabar_exists	Check if casa toolbar exists or not
clear_header	Clear header
create_mask	Interactively define a mask
figtext	Add text to figure at specified location
gca	Get current axes
plot	Plot a scantable
plot_lines	Plot a line catalog
plotazel	Plot azimuth and elevation versus time of a scantable
plotpointing	Plot telescope pointings in a scantable
plottp	Plot total power data
print_header	Print header of the scantable on the plot and/or logger
refresh	Do a soft refresh on the plot
save	Save the plot to a file
set_abcissa	Set the x-axis label of the plot
set_colors	Set the colors to be used
set_colours	Set the colors to be used
set_data	Set a scantable to plot
set_font	Set font properties
set_histogram	Enable/Disable histogram plotting
set_layout	Set the multi-panel layout

set_legend	Specify a mapping for the legend
set_linestyles	Set the linestyles to be used
set_mask	Set a plotting mask for a specific selection of the data
set_mode	Set the plots look and feel
set_ordinate	Set y-axis label of the plot
set_panelling	Set the panelling mode
set_range	Set the range of interest on the abcissa of the plot
set_selection	Set selection to the data
set_stacking	Set the stacking mode
set_title	Set the title of the plot
text	Add text in a specified location

sd.plotter.annotate.html

sd.plotter.annotate - Function

5.1.8 Annotate text at specified location

Description

Annotate text at specified location. This is an interface for matplotlib.axes.Axes.annotate function.
The interactive argument is specific to this method (not available from matplotlib). If it is True, you can set positions interactively using GUI panel.

Arguments

See matplotlib help about detailed description of arguments.

Inputs		
text	Annotate text	
	allowed:	string
	Default:	
xy	Position of the annotation	
	allowed:	float array
	Default:	
xytext	Position of the text	
	allowed:	float array
	Default:	None (use xy value)
xycoords	Coordinate of xy	
	allowed:	string
	Default:	'data'
textcoords	Coordinate of xytext	
	allowed:	string
	Default:	'data'
arrowprops	Line properties for the arrow	
	allowed:	dictionary
	Default:	None
interactive	Interactively set text position if True	
	allowed:	bool
	Default:	

Returns

Example

sd.plotter.arrow.html

sd.plotter.arrow - Function

5.1.8 Draw arrow on specified axis

Description

Draws arrow on specified axis from (x,y) to (x+dx,y+dy). This is an interface for matplotlib.axes.Axes.arrow function.
The interactive argument is specific to this method (not available from matplotlib). If it is True, you can set positions interactively using GUI panel.

Arguments

See matplotlib help about detailed description of arguments.

Inputs	
x	x position for the origin of the arrow
	allowed: float
	Default:
y	y position for the origin of the arrow
	allowed: float
	Default:
dx	x position for the terminal of the arrow (offset from origin)
	allowed: float
	Default:
dy	y position for the terminal of the arrow (offset from origin)
	allowed: float
	Default:
interactive	Interactively set origin and terminal of the arrow if True
	allowed: bool
	Default:

Returns

Example

sd.plotter.axhline.html

sd.plotter.axhline - Function

5.1.8 Draw a horizontal line

Description

Draw a horizontal line. This is an interface for matplotlib.axes.Axes.axhline function.
The interactive argument is specific to this method (not available from matplotlib). If it is True, you can set positions interactively using GUI panel.

Arguments

See matplotlib help about detailed description of arguments.

Inputs		
y	y position of the horizontal line	
	allowed:	float
	Default:	
xmin	Origin of the horizontal line	
	allowed:	float
	Default:	0
xmax	Terminate of the horizontal line	
	allowed:	float
	Default:	1
interactive	Interactively set origin and terminal of the line if True	
	allowed:	bool
	Default:	

Returns
Line2D object

Example

sd.plotter.axhspan.html

sd.plotter.axhspan - Function

5.1.8 Draw a horizontal rectangle

Description

Draw a horizontal rectangle. This is an interface for matplotlib.axes.Axes.axhspan function.
The interactive argument is specific to this method (not available from matplotlib). If it is True, you can set positions interactively using GUI panel.

Arguments

See matplotlib help about detailed description of arguments.

Inputs		
ymin	Bottom edge of the rectangle	
	allowed:	float
	Default:	
ymax	Top edge of the rectangle	
	allowed:	float
	Default:	
xmin	Left edge of the rectangle	
	allowed:	float
	Default:	0
xmax	Right edge of the rectangle	
	allowed:	float
	Default:	1
interactive	Interactively set the shape of the rectangle if True	
	allowed:	bool
	Default:	

Returns
Polygon object

Example

sd.plotter.axvline.html

sd.plotter.axvline - Function

5.1.8 Draw a vertical line

Description

Draw a vertical line. This is an interface for matplotlib.axes.Axes.axvline function.
The interactive argument is specific to this method (not available from matplotlib). If it is True, you can set positions interactively using GUI panel.

Arguments

See matplotlib help about detailed description of arguments.

Inputs		
x	x position of the vertical line	
	allowed:	float
	Default:	
ymin	Origin of the vertical line	
	allowed:	float
	Default:	0
ymax	Terminate of the vertical line	
	allowed:	float
	Default:	1
interactive	Interactively set origin and terminal of the line if True	
	allowed:	bool
	Default:	

Returns

Line2D object

Example

sd.plotter.axvspan.html

sd.plotter.axvspan - Function

5.1.8 Draw a vertical rectangle

Description

Draw a vertical rectangle. This is an interface for matplotlib.axes.Axes.axvspan function.
The interactive argument is specific to this method (not available from matplotlib). If it is True, you can set positions interactively using GUI panel.

Arguments

See matplotlib help about detailed description of arguments.

Inputs		
xmin	Left edge of the rectangle	
	allowed:	float
	Default:	
xmax	Right edge of the rectangle	
	allowed:	float
	Default:	
ymin	Bottom edge of the rectangle	
	allowed:	float
	Default:	0
ymax	Top edge of the rectangle	
	allowed:	float
	Default:	1
interactive	Interactively set the shape of the rectangle if True	
	allowed:	bool
	Default:	

Returns
Polygon object

Example

sd.plotter.casabar_exists.html

sd.plotter.casabar_exists - Function

5.1.8 Check if casa toolbar exists or not

Description

The function checks if plotter object associates with casa toolbar. It returns True if the object associates with casa toolbar. Otherwise, it returns False.

Arguments

Returns
bool

Example

sd.plotter.clear_header.html

sd.plotter.clear_header - Function

5.1.8 Clear header

Description

The function clears out header information from the plotter object. Nothing is done if plotter object doesn't have header.

Arguments

Returns

Example

sd.plotter.create_mask.html

sd.plotter.create_mask - Function

5.1.8 Interactively define a mask

Description

Interactively define a mask. It retruns a mask that is equivalent to the one created manually with scantable.create_mask.
The nwin argument indicates a number of mask windows to create interactively. The panel argument specifies which panel to use for mask selection (0-based). This is useful if different IFs are spread over panels.

Arguments

Inputs		
nwin	Number of mask windows to create interactively	
	allowed:	integer
	Default:	1
panel	Which panel to use for mask selection	
	allowed:	integer
	Default:	0

Returns
bool array

Example

sd.plotter.figtext.html

sd.plotter.figtext - Function

5.1.8 Add text to figure at specified location

Description

Add text to figure at location x,y (relative 0-1 coords). This method forwards *args and **kwargs to a Matplotlib method, matplotlib.Figure.text. See the method help for detailed information.

Arguments

Inputs		
x	x position of the text	
	allowed:	float
	Default:	
y	y position of the text	
	allowed:	float
	Default:	
s	Text to be added	
	allowed:	string
	Default:	

Returns

Example

sd.plotter.gca.html

sd.plotter.gca - Function

5.1.8 Get current axes

Description

Return the current axis object.

Arguments

Returns
Subplot object

Example

sd.plotter.plot.html

sd.plotter.plot - Function

5.1.8 Plot a scantable

Description

Plot a scantable.
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 abcissas of the scantable are consistent e.g. all 'channel' or all 'velocity' etc.

Arguments

Inputs		
scan	A scantable	
	allowed:	scantable
	Default:	None

Returns

Example

```
#create scantable
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
# plot
sd.plotter.plot(s)
```

sd.plotter.plot_lines.html

sd.plotter.plot_lines - Function

5.1.8 Plot a line catalog

Description

Plot a line catalog.
The linecat argument specifies actual catalog to be plot. It must be given as a linecatalog object.
Note that if the spectrum is flagged no line will be drawn in that location.

Arguments

Inputs		
linecat	The linecatalog to plot	
	allowed:	linecatalog
	Default:	None
doppler	The velocity shift to apply to the frequencies	
	allowed:	float
	Default:	0.0
deltachan	The number of channels to include each side of theline to determine a local maximum/minimum	
	allowed:	integer
	Default:	10
rotate	The rotation (in degrees) for the text label	
	allowed:	float
	Default:	90.0
location	The location of the line annotation from the 'top', 'bottom' or alternate	
	allowed:	string
	Default:	None

Returns

Example

sd.plotter.plotazel.html

sd.plotter.plotazel - Function

5.1.8 Plot azimuth and elevation versus time of a scantable

Description

Plot azimuth and elevation versus time of a scantable. If outfile is specified, the plot will be saved to a disk.

Arguments

Inputs		
scan	A scantable	
	allowed:	scantable
	Default:	None
outfile	Output file name	
	allowed:	string
	Default:	None

Returns

Example

sd.plotter.plotpointing.html

sd.plotter.plotpointing - Function

5.1.8 Plot telescope pointings in a scantable

Description

Plot telescope pointings in a scantable. If outfile is specified, the plot will be saved to a disk.

Arguments

Inputs		
scan	A scantable	
	allowed:	scantable
	Default:	None
outfile	Output file name	
	allowed:	string
	Default:	None

Returns

Example

sd.plotter.plottp.html

sd.plotter.plottp - Function

5.1.8 Plot total power data

Description

Plot total power data versus row number (or time). If outfile is specified, the plot will be saved to a disk.

Arguments

Inputs		
scan	A scantable	
	allowed:	scantable
	Default:	None
outfile	Output file name	
	allowed:	string
	Default:	None

Returns

Example

sd.plotter.print_header.html

sd.plotter.print_header - Function

5.1.8 Print header of the scantable on the plot and/or logger

Description

Print data (scantable) header on the plot and/or logger.

Arguments

Inputs		
plot	Whether or not print header info on the plot	
	allowed:	bool
	Default:	True
fontsize	Header font size (valid only plot=True)	
	allowed:	integer
	Default:	9
logger	Whether or not print header info on the logger	
	allowed:	bool
	Default:	False
selstr	Additional selection string (not verified)	
	allowed:	string
	Default:	”
extrastr	Additional string to print (not verified)	
	allowed:	string
	Default:	”

Returns

Example

sd.plotter.refresh.html

sd.plotter.refresh - Function

5.1.8 Do a soft refresh on the plot

Description

Do a soft refresh on the plot.

Arguments

Returns

Example

sd.plotter.save.html

sd.plotter.save - Function

5.1.8 Save the plot to a file

Description

Save the plot to a file. The known formats are 'png', 'ps', 'eps'.
If filename have suffix, the image format will be automatically detected. If no filename is specified a file called 'yyyymmdd_hhmmss.png' is created in the current directory.
The orientation argument is an 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.

Arguments

Inputs		
filename	The name of the output file	
	allowed:	bool
	Default:	True
orientation	Optional parameter for postscript that specifies orientation of the plot	
	allowed:	string ('landscape' or 'portrait')
	Default:	None
dpi	The dpi of the output non-postscript plot	
	allowed:	integer
	Default:	False (150 dpi)

Returns

Example

sd.plotter.set_abcissa.html

sd.plotter.set_abcissa - Function

5.1.8 Set the x-axis label of the plot

Description

Set the x-axis label of the plot. If multiple panels are plotted, multiple labels have to be specified.
If no abcissa labels are specified (i.e. None), data determine the labels.

Arguments

Inputs		
abcissa	A list of abcissa labels	
	allowed:	string, string array
	Default:	None
fontsize	A font size of the label	
	allowed:	integer
	Default:	None
refresh	If True, the plot is replotted based on the new parameter setting(s)	
	allowed:	bool
	Default:	True

Returns

Example

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

sd.plotter.set_colors.html

sd.plotter.set_colors - Function

5.1.8 Set the colors to be used

Description

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

Arguments

Inputs		
colmap	A list of color names	
	allowed:	string
	Default:	
refresh	If True, the plot is replotted based on the new parameter setting(s)	
	allowed:	bool
	Default:	True

Returns

Example

```
sd.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.
```

sd.plotter.set_colours.html

sd.plotter.set_colours - Function

5.1.8 Set the colors to be used

Description

See set_colors.

Arguments

Returns

Example

sd.plotter.set_data.html

sd.plotter.set_data - Function

5.1.8 Set a scantable to plot

Description

Set a scantable to plot.
Note that the user specified masks and data selections will be reset if a new scantable is set. This method should be called before setting data selections (set_selection) and/or masks (set_mask).

Arguments

Inputs		
scan	A scantable	
	allowed:	scantable
	Default:	
refresh	If True, the plot is replotted based on the new parameter setting(s)	
	allowed:	bool
	Default:	True

Returns

Example

sd.plotter.set_font.html

sd.plotter.set_font - Function

5.1.8 Set font properties

Description

Set font properties.

Arguments

Inputs	
family	One of 'sans-serif', 'serif', 'cursive', 'fantasy', 'monospace' allowed: string Default:
style	One of 'normal' (or 'roman'), 'italic' or 'oblique' allowed: string Default:
weight	One of 'normal or 'bold' allowed: string Default:
size	the 'general' font size, individual elements can be adjusted seperately allowed: integer Default:
refresh	If True, the plot is replotted based on the new parameter setting(s) allowed: bool Default: True

Returns

Example

sd.plotter.set_histogram.html

sd.plotter.set_histogram - Function

5.1.8 Enable/Disable histogram plotting

Description

Enable/Disable histogram-like plotting.
The first default is taken from sd.rcParams[plotter.histogram].

Arguments

Inputs		
hist	Enable/Disable histogram plotting	
	allowed:	bool
	Default:	True
refresh	If True, the plot is replotted based on the new parameter setting(s)	
	allowed:	bool
	Default:	True

Returns

Example

sd.plotter.set_layout.html

sd.plotter.set_layout - Function

5.1.8 Set the multi-panel layout

Description

Set the multi-panel layout, i.e. how many rows and columns plots are visible.
Note that if no argument is given, the potter reverts to its auto-plot behaviour.

Arguments

Inputs		
rows	Number of rows of plots	
	allowed:	integer
	Default:	None
cols	Number of columns of plots	
	allowed:	integer
	Default:	None
refresh	If True, the plot is replotted based on the new parameter setting(s)	
	allowed:	bool
	Default:	True

Returns

Example

sd.plotter.set_legend.html

sd.plotter.set_legend - Function

5.1.8 Specify a mapping for the legend

Description

Specify a mapping for the legend instead of using the default indices.
The list of legends should be given to the mp argument as a string. 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'$x^{2}$'`
The mode argument controls where to display the legend. It should be specified as an integer. The following list shows a meaning of each integer.

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

Arguments

Inputs		
mp	A list of legend strings	
	allowed:	string array
	Default:	None
fontsize	The font size of the label	
	allowed:	integer
	Default:	None
mode	Where to display the legend	
	allowed:	integer
	Default:	0 (auto)
refresh	If True, the plot is replotted based on the new parameter setting(s)	
	allowed:	bool
	Default:	True

Returns

Example

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

sd.plotter.set_linestyles.html

sd.plotter.set_linestyles - Function

5.1.8 Set the linestyles to be used

Description

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. Accepted linestyles are 'line', 'dashed', 'dotted', 'dashdot', 'dashdotdot' and 'dashdashdot'.

Arguments

Inputs		
linestyles	A list of linestyles to use	
	allowed:	string array
	Default:	None
linewidth	A width of the line	
	allowed:	integer
	Default:	None
refresh	If True, the plot is replotted based on the new parameter setting(s)	
	allowed:	bool
	Default:	True

Returns

Example

```
sd.plotter.set_colors("black")
sd.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'.
```

sd.plotter.set_mask.html

sd.plotter.set_mask - Function

5.1.8 Set a plotting mask for a specific selection of the data

Description

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

Arguments

Inputs		
mask	A mask from scantable.create_mask	
	allowed:	bool array
	Default:	None
selection	The spectra to apply the mask to	
	allowed:	selector
	Default:	None
refresh	If True, the plot is replotted based on the new parameter setting(s)	
	allowed:	bool
	Default:	True

Returns

Example

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

sd.plotter.set_mode.html

sd.plotter.set_mode - Function

5.1.8 Set the plots look and feel

Description

Set the plots look and feel, i.e. what you want to see on the plot. Stacking is a plot as line color overlays, while panelling is a plot across multiple panels. By default, the stacking is set to 'pol' and the panelling is set to 'scan'.
Valid modes are:

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

Arguments

Inputs		
stacking	Tell the plotter which variable to plot as line colour overlays	
	allowed:	string
	Default:	None
panelling	Tell the plotter which variable to plot across multiple panels	
	allowed:	string
	Default:	None
refresh	If True, the plot is replotted based on the new parameter setting(s)	
	allowed:	bool
	Default:	True

Returns

Example

sd.plotter.set_ordinate.html

sd.plotter.set_ordinate - Function

5.1.8 Set y-axis label of the plot

Description

Set the y-axis label of the plot. If multiple panels are plotted, multiple labels have to be specified. If no ordinate is set, data determine the lebels.

Arguments

Inputs		
ordinate	A list of ordinate labels	
	allowed:	string, string array
	Default:	None
fontsize	A font size of the label	
	allowed:	integer
	Default:	None
refresh	If True, the plot is replotted based on the new parameter setting(s)	
	allowed:	bool
	Default:	True

Returns

Example

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

sd.plotter.set_panelling.html

sd.plotter.set_panelling - Function

5.1.8 Set the panelling mode

Description

Set the 'panelling' mode i.e. which type of spectra should be spread across different panels.
Valid modes are:

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

Arguments

Inputs		
what	Which type of spectra should be spread across different panels	
	allowed:	string
	Default:	None

Returns

Example

sd.plotter.set_range.html

sd.plotter.set_range - Function

5.1.8 Set the range of interest on the abcissa of the plot

Description

Set the range of interest on the abcissa of the plot.
These become non-sensical when the unit changes. In that case, use plotter.set_range() without parameters to reset.

Arguments

Inputs	
xstart	The start point of the 'zoom' window
	allowed: float
	Default: None
xend	The end point of the 'zoom' window
	allowed: float
	Default: None
ystart	The start point of the 'zoom' window
	allowed: float
	Default: None
yend	The end point of the 'zoom' window
	allowed: float
	Default: None
refresh	If True, the plot is replotted based on the new parameter setting(s)
	allowed: bool
	Default: True
offset	Shift the abcissa by the given amount. The abcissa label will have '(relative)' appended to it.
	allowed: float
	Default: True

Returns

Example

sd.plotter.set_selection.html

sd.plotter.set_selection - Function

5.1.8 Set selection to the data

Description

Set selection to the data. The method allows both to set selector object directly and to pass arguments for constructor of the selector to create new selector object internally.
See selector for valid arguments for constructor of the selector.

Arguments

Inputs		
selection	A selector object	
	allowed:	selector
	Default:	None
refresh	If True, the plot is replotted based on the new parameter setting(s)	
	allowed:	bool
	Default:	True

Returns

Example

```
# these are equivalent
sel=sd.selector(ifs=[0])
sd.plotter.set_selection(sel)
sd.plotter.set_selection(ifs=[0])
```

sd.plotter.set_stacking.html

sd.plotter.set_stacking - Function

5.1.8 Set the stacking mode

Description

Set the 'stacking' mode i.e. which type of spectra should be overlayed.
Valid modes are:

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

Arguments

Inputs		
what	Which type of spectra should be overlayed	
	allowed:	string
	Default:	None

Returns

Example

sd.plotter.set_title.html

sd.plotter.set_title - Function

5.1.8 Set the title of the plot

Description

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

Arguments

Inputs		
title	The title	
	allowed:	string, string array
	Default:	None
fontsize	A font size of the title	
	allowed:	integer
	Default:	None
refresh	If True, the plot is replotted based on the new parameter setting(s)	
	allowed:	bool
	Default:	True

Returns

Example

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

sd.plotter.text.html

sd.plotter.text - Function

5.1.8 Add text in a specified location

Description

Add text in string s to axis at location x,y (data coords). This is an interface for matplotlib.axes.Axes.text function.
The interactive argument is specific to this method (not available from matplotlib). If it is True, you can set positions interactively using GUI panel.

Arguments

See matplotlib help about detailed description of arguments.

Inputs		
x	x position of the text	
	allowed:	float
	Default:	
y	y position of the text	
	allowed:	float
	Default:	
s	The text to be drawn	
	allowed:	string
	Default:	
interactive	Interactively set the position of the text if True	
	allowed:	bool
	Default:	

Returns

Example

sd.coordinate-Tool.html

5.1.9 sd.coordinate - Tool

Single-dish specific spectral coordinate conversion

Description

The coordinate class is a representation of the spectral coordinate (frequency axis) of the data. It handles a conversion between pixel/channel values and frequency/velocity ones under the current spectral coordinate.
Spectral coordinate is composed of a set of three values: a reference pixel, a frequency in Hz at the reference pixel, and an increment in Hz (width of each pixel).
Normally, the object of this class is not created from scratch. Instead, the object can be obtained from scantable object using get_coordinate method. Returned spectral coordinate contains spectral coordinate information and rest frequency value, which is needed to handle velocity conversion, of the given row.

Example

```
# create a scantable object
s=sd.scantable('OrionS_rawACSmod',average=False)
# get coordinate system
c=s.get_coordinate(0)
# get coordinate system values
c.get_increment()
6104.2329788208008
c.get_reference_pixel()
4096.0
c.get_reference_value()
45489353563.344795
# get pixel value
c.to_pixel(4.5489e10)
4038.0789891175605
# get frequency at channel 0
c.to_frequency(0)
45464350625.063545
# get velocity at channel 0
c.to_velocity(0)
170.73624010940924
```

Methods

coordinate	Constructor
coordinate.get_increment	Get increment of this coordinate system
coordinate.get_reference_pixel	Get reference pixel of this coordinate system
coordinate.get_reference_value	Get reference value of this coordinate system
coordinate.to_frequency	Convert a channel/pixel value to a frequency
coordinate.to_pixel	Convert a frequency value to a channel/pixel
coordinate.to_velocity	Convert a channel/pixel value to a velocity

sd.coordinate.coordinate.html

sd.coordinate.coordinate - Function

5.1.9 Constructor

Description

Constructor of this class. Not useful any more.

Arguments

Returns

Example

sd.coordinate.coordinate.get_increment.html

sd.coordinate.coordinate.get_increment - Function

5.1.9 Get increment of this coordinate system

Description

Return an increment of this spectral coordinate. The unit is Hz.

Arguments

Returns
float

Example

sd.coordinate.coordinate.get_reference_pixel.html

sd.coordinate.coordinate.get_reference_pixel - Function

5.1.9 Get reference pixel of this coordinate system

Description

Return a reference pixel of this spectral coordinate.

Arguments

Returns
float

Example

sd.coordinate.coordinate.get_reference_value.html

sd.coordinate.coordinate.get_reference_value - Function

5.1.9 Get reference value of this coordinate system

Description

Return a reference frequency of this spectral coordinate. The unit is Hz.

Arguments

Returns

Example

sd.coordinate.coordinate.to_frequency.html

sd.coordinate.coordinate.to_frequency - Function

5.1.9 Convert a channel/pixel value to a frequency

Description

The method converts a given channel/pixel value to a frequency under this spectral coordinate. Default unit of returned value is Hz.

Arguments

Inputs		
pixel	Channel/pixel value where frequency wants to know	
	allowed:	float
	Default:	
unit	Unit of the returned value	
	allowed:	string ('Hz', 'kHz', 'MHz', 'GHz')
	Default:	'Hz'

Returns
float

Example

sd.coordinate.coordinate.to_pixel.html

sd.coordinate.coordinate.to_pixel - Function

5.1.9 Convert a frequency value to a channel/pixel

Description

The method converts a given frequency value to a channel/pixel under this spectral coordinate.

Arguments

Inputs	
	A frequency value that want to convert to channel/pixel
	allowed: float
	Default:

Returns
float

Example

sd.coordinate.coordinate.to_velocity.html

sd.coordinate.coordinate.to_velocity - Function

5.1.9 Convert a channel/pixel value to a velocity

Description

The method converts a given channel/pixel value to a velocity under this spectral coordinate. Default unit of returned value is km/s.

Arguments

Inputs		
pixel	Channel/pixel value where velocity wants to know	
	allowed:	float
	Default:	
unit	Unit of the returned value	
	allowed:	string ('km/s', 'm/s')
	Default:	'km/s'

Returns
float

Example

sd.opacity_model.html

5.1.10 sd.opacity_model - Tool

Single dish specific opacity model

Description

This class implements opacity/atmospheric brightness temperature model equivalent to the model available in MIRIAD. The actual math is a convertion of the Fortran code written by Bob Sault for MIRIAD. It implements a simple model of the atmosphere and Liebe's model (1985) of the complex refractive index of air.
The model of the atmosphere is one with an exponential fall-off in the water vapour content (scale height of 1540 m) and a temperature lapse rate of 6.5 mK/m. Otherwise the atmosphere obeys the ideal gas equation and hydrostatic equilibrium.
Note that the model includes atmospheric lines up to 800 GHz, but was not rigorously tested above 100 GHz and for instruments located at a significant elevation. For high-elevation sites it may be necessary to adjust scale height and lapse rate.
The constructor takes several arguments that specifies observatory and weather informations.

Arguments

Inputs	
temperature	Air temperature at the observatory (K) allowed: float Default: 288.0
pressure	Air pressure at the sea level if the observatory elevation is set to non-zero value (note, by default is set to 700m) or at the observatory ground level if the elevation is set to 0. The value is in Pascals or hPa. allowed: float Default: 101325.0
humidity	Air humidity at the observatory (fractional) allowed: float Default: 0.5
elevation	Observatory elevation about sea level (in meters) allowed: float Default: 700.0

Example

```
o=sd.opacity_model(temperature=300.0,elevation=1300.0)
tau=sd.get_opacities([1.0e11,1.1e11])
print tau
[0.2363221976673901, 0.3068089293871521]
```

Methods

get_opacities	Get the opacity value(s) for the given frequency(ies)
set_observatory_elevation	Update the model using the given observatory elevation
set_weather	Update the model using the given environmental parameters

sd.opacity_model.get_opacities.html

sd.opacity_model.get_opacities - Function

5.1.10 Get the opacity value(s) for the given frequency(ies)

Description

Get the opacity value(s) for the given frequency(ies). If no elevation is given the opacities for the zenith are returned. If an elevation is specified refraction is also taken into account. The user is able to set frequency value(s) where the user want to compute opacity(ies). One opacity value per frequency is returned as a scalar or list.

Arguments

Inputs		
freq	A frequency value in Hz, or a list of frequency values.	
	allowed:	float, float array
	Default:	
elevation	The elevation in radian at which to compute the opacity.	
	allowed:	float
	Default:	None (zenith opacity)

Returns

float, float array

Example

sd.opacity_model.set_observatory_elevation.html

sd.opacity_model.set_observatory_elevation - Function

5.1.10 Update the model using the given observatory elevation

Description

Update the model using the given the observatory elevation in meter. Note that, in constructor, the default value for the observatory elevation is 700m.

Arguments

Inputs	
elevation	The site elevation in meter at which to compute the opacity. allowed: float Default:

Returns

Example

sd.opacity_model.set_weather.html

sd.opacity_model.set_weather - Function

5.1.10 Update the model using the given environmental parameters

Description

Update the model using the given environmental parameters. The pressure value will be a pressure at sea level is the observatory elevation is set to non-zero value, while it will be a pressure at the observatory ground level if the elevation is set to 0.

Arguments

Inputs	
temperature	Air temperature at the observatory (K)
	allowed: float
	Default:
pressure	Air pressure in Pascals or hPa.
	allowed: float
	Default:
humidity	Air humidity at the observatory (fractional)
	allowed: float
	Default:

Returns

Example

sd.asapgrid-Tool.html

5.1.11 sd.asapgrid - Tool

Tool to convolve map data onto regularly spaced grid

Description

The asapgrid class is defined to convolve data onto regular spatial grid.

Arguments

Inputs		
infile	input data as a string or string list	
	allowed:	string, list of string
	Default:	

Returns

asapgrid instance

Example

```
# create asapgrid instance with two input data
g = asapgrid( ['testimage1.asap','testimage2.asap'] )
# set IFNO if necessary
g.setIF( 0 )
# set POLNOs if necessary
g.setPolList( [0,1] )
# set SCANNOs if necessary
g.setScanList( [22,23,24] )
# define image with full specification
# you can skip some parameters (see help for defineImage)
g.defineImage( nx=12, ny=12, cellx='10arcsec', celly='10arcsec',
               center='J2000 10h10m10s -5d05m05s' )
# set convolution function
g.setFunc( func='sf', width=3 )
# enable min/max clipping
g.enableClip()
# or, disable min/max clipping
#g.disableClip()
# actual gridding
```

```
g.grid()
# save result
g.save( outfile='grid.asap' )
# plot result
g.plot( plotchan=1246, plotpol=-1, plotgrid=True, plotobs=True )
```

Methods

sd.asapgrid.defineImage	Define spatial grid
sd.asapgrid.disableClip	Disable min/max clipping
sd.asapgrid.enableClip	Enable min/max clipping
sd.asapgrid.grid	Do gridding
sd.asapgrid.plot	Plot result
sd.asapgrid.save	Save result
sd.asapgrid.setData	Set data to be processed
sd.asapgrid.setFunc	Set convolution function
sd.asapgrid.setIF	Set IFNO to be processed
sd.asapgrid.setPolList	Set polarizations to be processed
sd.asapgrid.setScanList	Set scans to be processed
sd.asapgrid.setWeight	Set weight type

sd.asapgrid.defineImage.html

sd.asapgrid.defineImage - Function

5.1.11 Define spatial grid

Description

Define spatial grid.
First two parameters, nx and ny, define number of pixels of the grid. If which of those is not specified, it will be set to the same value as the other. If none of them are specified, it will be determined from map extent and cell size.
Next two parameters, cellx and celly, define size of pixel. You should set those parameters as string, which is constructed numerical value and unit, e.g. '0.5arcmin', or numerical value. If those values are specified as numerical value, their units will be assumed to 'arcmin'. If which of those is not specified, it will be set to the same value as the other. If none of them are specified, it will be determined from map extent and number of pixels, or set to '1arcmin' if neither nx nor ny is set.
The last parameter, center, define the central coordinate of the grid. You should specify its value as a string, like,
'J2000 05h08m50s -16d23m30s'
or
'J2000 05:08:50 -16.23.30'
You can omit equinox when you specify center coordinate. In that case, J2000 is assumed. If center is not specified, it will be determined from the observed positions of input data.

Arguments

Inputs		
nx	number of pixels along x (R.A.) direction	
	allowed:	int
	Default:	-1
ny	number of pixels along y (Dec.) direction	
	allowed:	int
	Default:	-1
cellx	size of pixel in x (R.A.) direction	
	allowed:	string, float
	Default:	”
celly	size of pixel in y (Dec.) direction.	
	allowed:	string, float
	Default:	”
center	central position of the grid	
	allowed:	string
	Default:	”

Returns

Example

sd.asapgrid.disableClip.html

sd.asapgrid.disableClip - Function

5.1.11 Disable min/max clipping

Description

Disable min/max clipping.

Arguments

Returns

Example

sd.asapgrid.enableClip.html

sd.asapgrid.enableClip - Function

5.1.11 Enable min/max clipping

Description

Enable min/max clipping.

Arguments

Returns

Example

sd.asapgrid.grid.html

sd.asapgrid.grid - Function

5.1.11 Do gridding

Description

Actual gridding which will be done based on several user inputs.

Arguments

Returns

Example

sd.asapgrid.plot.html

sd.asapgrid.plot - Function

5.1.11 Plot result

Description

Plot gridded data. Data must be saved using save method.

Arguments

Inputs	
plotchan	Which channel you want to plot. Default is to average all the channels. allowed: int Default: -1
plotpol	Which polarization component you want to plot. Default is to average all the polarization components. allowed: int Default: -1
plotobs	Also plot observed position if True. Setting True for large amount of spectra might be time consuming. allowed: bool Default: False
plotgrid	Also plot grid center if True. Setting True for large number of grids might be time consuming. allowed: bool Default: False

Returns

Example

sd.asapgrid.save.html

sd.asapgrid.save - Function

5.1.11 Save result

Description

Save result. By default, output data name will be constructed from first element of input data name list (e.g. 'input.asap.grid').

Arguments

Inputs		
outfile	output data name	
	allowed:	string
	Default:	”

Returns

Example

sd.asapgrid.setData.html

sd.asapgrid.setData - Function

5.1.11 Set data to be processed

Description

Set data to be processed.

Arguments

Inputs	
infile	input data as a string or string list if you want to grid more than one data at once. allowed: string, string list Default: ”

Returns

Example

sd.asapgrid.setFunc.html

sd.asapgrid.setFunc - Function

5.1.11 Set convolution function

Description

Set convolution function. Possible options are 'box' (Box-car, default), 'sf' (prolate spheroidal), and 'gauss' (Gaussian). Width of convolution function can be set using width parameter. By default (-1), width is automatically set depending on each convolution function. Default values for width are: 'box': 1 pixel 'sf': 3 pixels 'gauss': 1 pixel (width is used as HWHM)

Arguments

Inputs		
func	Function type ('box', 'sf', 'gauss')	
	allowed:	string
	Default:	box
width	Width of convolution function. Default (-1) is to choose pre-defined value for each convolution function.	
	allowed:	int
	Default:	-1

Returns

Example

sd.asapgrid.setIF.html

sd.asapgrid.setIF - Function

5.1.11 Set IFNO to be processed

Description

Set IFNO to be processed. Currently, asapgrid allows to process only one IFNO for one gridding run even if the data contains multiple IFs. If you didn't specify IFNO, default value, which is IFNO in the first spectrum, will be processed.

Arguments

Inputs		
ifno	IFNO to be processed	
	allowed:	int
	Default:	

Returns

Example

sd.asapgrid.setPolList.html

sd.asapgrid.setPolList - Function

5.1.11 Set polarizations to be processed

Description

Set list of polarization components you want to process. If not specified, all POLNOs will be processed.

Arguments

Inputs		
pollist	list of POLNOs to be processed	
	allowed:	int list
	Default:	

Returns

Example

sd.asapgrid.setScanList.html

sd.asapgrid.setScanList - Function

5.1.11 Set scans to be processed

Description

Set list of scans you want to process. If not specified, all scans will be processed.

Arguments

Inputs		
scanlist	list of SCANNOs to be processed	
	allowed:	int list
	Default:	

Returns

Example

sd.asapgrid.setWeight.html

sd.asapgrid.setWeight - Function

5.1.11 Set weight type

Description

Set weight type. Possible options are 'uniform' (default), 'tint' (weight by integration time), 'tsys' (weight by Tsys: 1/Tsys**2), and 'tintsys' (weight by integration time as well as Tsys: tint/Tsys**2).

Arguments

Inputs		
weightType	weight type ('uniform', 'tint', 'tsys', 'tintsys')	
	allowed:	string
	Default:	uniform

Returns

Example

sd.asaplog-Tool.html

5.1.12 sd.asaplog - Tool

Wrapper object to allow for both casapy and asap logging

Description

Wrapper object to allow for both casapy and asap logging.
Inside casapy this will connect to 'taskinit.casalog'. Otherwise it will create its own casa log sink.
.. note:: Do not instantiate a new one - use the :obj:'asaplog' instead.
In the ASAP logging system, log messages are accumulated to the log buffer when you post any message using sd.asaplog.push. Buffered messages are flushed when you call sd.asaplog.post.
Methods

sd.asaplog.clear	Clear buffer
sd.asaplog.disable	Disable (or enable) logging
sd.asaplog.enable	Enable (or disable) logging
sd.asaplog.is_enabled	Query if logging is enabled
sd.asaplog.post	Post message to the logger
sd.asaplog.push	Push logs into the buffer

sd.asaplog.clear.html

sd.asaplog.clear - Function

5.1.12 Clear buffer

Description

Clear buffer. This is only effective for standalone ASAP.

Arguments

Returns

Example

sd.asaplog.disable.html

sd.asaplog.disable - Function

5.1.12 Disable (or enable) logging

Description

Disable (or enable) logging.

Arguments

Inputs		
flag	Set False to disable logging	
	allowed:	bool
	Default:	False

Returns

Example

sd.asaplog.enable.html

sd.asaplog.enable - Function

5.1.12 Enable (or disable) logging

Description

Enable (or disable) logging.

Arguments

Inputs		
flag	Set True to enable logging	
	allowed:	bool
	Default:	True

Returns

Example

sd.asaplog.is_enabled.html

sd.asaplog.is_enabled - Function

5.1.12 Query if logging is enabled

Description

Query if logging is enabled.

Arguments

Returns

Example
bool

sd.asaplog.post.html

sd.asaplog.post - Function

5.1.12 Post message to the logger

Description

Post the messages to the logger. This will clear the buffered logs.

Arguments

Inputs		
level	The log level (severity). One of INFO, WARN, ERROR.	
	allowed:	string
	Default:	INFO
origin	Origin of the log message.	
	allowed:	string
	Default:	

Returns

Example

sd.asaplog.push.html

sd.asaplog.push - Function

5.1.12 Push logs into the buffer

Description

Push logs into the buffer. post needs to be called to send them.

Arguments

Inputs		
msg	the log message	
	allowed:	string
	Default:	
newline	should we terminate with a newline	
	allowed:	bool
	Default:	True

Returns

Example

Permissions

- Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.
- Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.
- Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by AUI.