1.5 qimager - Module

Module for synthesis and single dish imaging
include "qimager.g"

Description

qimager is a stripped down version of imager for experimentation
with processing algorithms.
New experimental capabilities:

1. W projection - a new approach to non-coplanar baseslines.

Tools

qimager	tool for synthesis imaging

Functions

qimageralltests	Perform all qimager tests using standard MeasurementSets
qimagercomponenttest	Perform a test of component model processing using a standard MeasurementSet
qimagerlongtest	Perform a long test of imaging from a standard MeasurementSet
qimagermaketestcl	Make a standard componentlist for a standard MeasurementSet
qimagermaketestmfms	Make a standard multifield MeasurementSet that can be used for experimentation
qimagermaketestms	Make a standard MeasurementSet that can be used for experimentation
qimagermaketestsdms	Make a standard single dish MeasurementSet that can be used for experimentation
qimagermftest	Perform a test of multi-field processing using a standard MeasurementSet
qimagermultiscale	Makes multi-resolution multiscale images.
qimagerpbtest	Perform a test of primary beam processing using a standard MeasurementSet
qimagersdtest	Perform a test of single dish imaging from a standard MeasurementSet
qimagerselfcaltest	Perform a test of selfcal processing using a standard MeasurementSet
qimagerspectraltest	Perform a test of spectral processing using a standard spectral line MeasurementSet
qimagertest	Perform a test of imaging from a standard MeasurementSet

1.5.1 qimager - Tool

Package	synthesis
Module	qimager

tool for synthesis imaging

Description
qimager is an tool that accomplishes synthesis processing. A

qimager must be constructed for each MeasurementSet for which one wishes to do processing. Multiple copies of qimager may be made at any time (provide they are given different names).

Constructors

qimager	Construct an qimager tool

Functions

advise	Advise (and optionally use) parameter values
approximatepsf	Calculate approximate point spread functions
clean	Calculate a deconvolved image with selected clean algorithm
close	Close the qimager tool, with data written on disk, keeping qimager process running for future use
done	Terminate the qimager process
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
linearmosaic	Make a linear mosaic of several images
make	Make an empty (i.e. blank) image
makeimage	Calculate images by gridding, etc.
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
predict	Predict the data from the specified model and componentlist
regionmask	Construct a mask image from a region
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
selfcal	Self-calibrate the MeasurementSet using a calibrater tool and model+component list
sensitivity	Calculate rms sensitivity
setbeam	Set the beam parameters for clean restoration
setdata	Set the data parameters selection for subsequent processing
setimage	Set the image parameters for subsequent processing
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
setsdoptions	Set some options for single dish processing
setvp	Set the voltage pattern model for subsequent processing
smooth	Calculate an image smoothed with a Gaussian beam
stop	stop the currently executing function asap
summary	Summarize the current state of the qimager tool
uvrange	Select data within the limit of a given range
weight	Apply additional weighting to the visibility weights

qimager - Constructor

Package	synthesis
Module	qimager
Tool	qimager

Construct an qimager tool

Synopsis

qimager(filename, compress, host, forcenewserver)

Description

This is used to construct qimager tools associated with a MeasurementSet. The qimager 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 qimager.

Arguments

filename	MeasurementSet to be imaged Allowed: Table name Default: ''
compress	Compress calibration columns? Allowed: Bool Default: F
host	Host on which to run qimager Allowed: String: '' means localhost Default: ''
forcenewserver	Flag to force a new qimager client Allowed: Bool Default: T

Returns

qimager tool

Example

```
imgr:=qimager('3C273XC1.MS')
```

```
imgr.setimage(nx=256, ny=256, cellx='0.7arcsec', celly='0.7arcsec')
imgr.image(type='corrected', image='3C273XC1.dirty')
imgr.close()
```

---

qimager.open - Function

Package	synthesis
Module	qimager
Tool	qimager

Open a new MeasurementSet, for processing, closing current MeasurementSet

Synopsis
open(thems, compress)

Description
Close the current MeasurementSet and open a new MeasurementSet instead. The current state of qimager is retained, except for the data selection.

Arguments

thems	New MeasurementSet to be processed AllowedTable name Default:'MS'
compress	Compress calibration columns? AllowedBool DefaultF

Returns
Bool

---

qimager.close - Function

Package	synthesis
Module	qimager
Tool	qimager

Close the qimager tool, with data written on disk, keeping qimager process running for future use

Synopsis
close()

Description
This is used to close qimager tools. Note that the data is written to disk. The qimager process keeps running until a done tool function call is performed.

Returns
Bool

Example

```
imgr:=qimager('3C273XC1.MS')
imgr.image(image='3C273XC1.dirty',type='corrected')
imgr.close()
```

---

qimager.stop - Function

Package	synthesis
Module	qimager
Tool	qimager

stop the currently executing function asap

Synopsis
stop()

Description
Stop the currently executing function as soon as possible. Note that it is not always possible to stop a function.

Returns
Bool

---

qimager.done - Function

Package	synthesis
Module	qimager
Tool	qimager

Terminate the qimager process

Synopsis
done()

Description
This is used to totally stop the qimager process. It is a good idea to conserve memory use on your machine by stopping the process once you no longer need it.

Returns
Bool

Example

```
imgr:=qimager('3C273XC1.MS')
imgr.image(image='3C273XC1.dirty',type='corrected')
imgr.close()
imgr.done()
```

---

qimager.summary - Function

Package	synthesis
Module	qimager
Tool	qimager

Summarize the current state of the qimager tool

Synopsis
summary()

Description
Writes a summary of the properties of the qimager 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 setimage)
- The current beam (set by fitpsf or setbeam.
- The selection of an ms (set via setdata)
- The general processing options (set via setoptions)

Returns
Bool

Example

```
- imgr:=qimager('3C273XC1.MS');
- imgr.setimage(npix=[256,256])
- imgr.summary()
```

---

qimager.setdata - Function

Package	synthesis
Module	qimager
Tool	qimager

Set the data parameters selection for subsequent processing

Synopsis
setdata(mode, nchan, start, step, mstart, mstep, spwid, fieldid, msselect, async)

Description
This setup tool function selects which data are to be used subsequently. After invocation of setdata, 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 the imaging, and may or may not be imaged, depending on how the image was constructed using setimage. For example, in mosaicing, use fieldid in setimage to control what pointing is used to define the field center, and use fieldid in setdata 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 is terse till a better selection scheme is devised.

The selection is controlled by the mode argument:

channel Selection in channels using the nchan, start and step arguments

velocity Selection in radio velocity, using the nchan, mstart and mstep arguments.

opticalvelocity As for velocity but the velocity definition is optical.

For channel mode, the other fields have the following meaning:

nchan is the number of output channels produced. Generally it defaults to the number of input channels.

start is the first channel from input dataset that is to be used. It defaults to 1 (i.e. first channel).

step gives the increment between selected input channels.

The channels are centered on velocities: start, start+step, start+2*step, etc.

By choosing the parameters for setdata and setimage 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):

```
imgr.setimage(mode='channel', nchan=64, start=1, step=8);
imgr.setdata(mode='channel', nchan=512, start=1, step=1)
imgr.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:

```
imgr.setdata(mode='channel', nchan=64, start=1, step=8)
imgr.setimage(mode='channel', nchan=64, start=1, step=8);
imgr.clean();
```

For velocity and opticalvelocity modes, the mstart and mstep are the start and step velocities as strings.

```
imgr.setimage(mode='velocity', nchan=64, mstart='20km/s', mstep='-100m/s');
imgr.setdata(mode='velocity', nchan=64, mstart='20km/s', mstep='-100m/s');
imgr.clean();
```

If the image and data selections differ, then averaging is done during the gridding and degridding process in the image deconvolution.

```
imgr.setimage(mode='channel', nchan=64, start=1, step=8);
imgr.setdata(mode='channel', nchan=512, start=1, step=1)
imgr.clean()
```

Note: The channels numbers used in setimage and setdata refers to the same channel. So if a channel is not selected in setdata but is selected in setimage, then blank channels image are made. The example below will result in the having the first 5 channels in the image to be blank.

```
imgr.setdata(mode='channel', nchan=50, start=6, step=1) #selected chan 6-55
imgr.setimage(mode='channel', nchan=50, start=1, 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
imgr.clean()
```

For multi-spectral window cube imaging the selection of the data can be done as follows

```
imgr.setdata(mode='channel', nchan=[50,60], start=[1,1], step=[1,1],
             spwid=[1,2])
imgr.setimage(mode='channel', nchan=110, start=1, step=1, spwid=[1,2]);
```

The above means that you would make a data selection of 50 channels (starting from 1 steping 1) from the first spectral window and 60 channels (starting from 1 steping 1). The setimage 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).

```
imgr.setdata(mode='channel', nchan=[50,60], start=[1,1], step=[1,1],
            spwid=[1,2])
imgr.setimage(mode='velocity', nchan=200, mstart='20km/s',
            mstep='-100m/s');
```

Arguments

mode	Type of processing: channel or velocity Allowed:'channel'|'velocity'
nchan	Number of channels to select Allowed:Vector of Ints Default:1
start	Start channels (1-relative) Allowed:Vector of Ints
step	Step in channel number Allowed:Vector of Int
mstart	Start velocity (e.g. '20Km/s') Allowed:Quantity
mstep	Step in velocity (e.g. '100m/s' Allowed:Quantity
spwid	Spectral Window Ids (1 relative) to select Allowed:Vector of Ints Default:1
fieldid	Field Ids (1 relative) to select Allowed:Vector of Ints Default:1
msselect	TQL select string applied as a logical "and" with the other selections Allowed:String
async	Run asynchronously in the background? Allowed:Bool Default:!dowait

Example

```
imgr:=qimager('3C273XC1.MS');
imgr.setdata(nchan=512,start=1,step=1, msselect='SCAN_NUMBER > 10 && FIELD_ID==2)
```

---

qimager.setimage - Function

Package	synthesis
Module	qimager
Tool	qimager

Set the image parameters for subsequent processing

Synopsis

setimage(nx, ny, cellx, celly, stokes, doshift, phasecenter, shiftx, shifty, mode, nchan, start, step, mstart, mstep, spwid, fieldid, facets, distance)

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 meaning of arguments mode, nchan, step, *etc.* is described in setdata. qimager can perform multi-frequency synthesis over several spectral windows (mode='mfs'). To acheive this, you should set spwid to an array of the required spectral windows (*e.g.* spwid=1:2).

WARNING: For multifrequency synthesis, 'mfs', it is important that the spwid's selected in setdata be the SAME as the one selected in 'setimage'. 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.

The phase center of the image defaults to that of the specified fieldid (the first if none is specified). This is important if you have multiple pointings in the data. The user would have used setdata to select which pointings would be used in imaging. Note that the fieldid refers to the ordering of fields in a MeasurementSet, and has no connection with the number of facets in an image. A phase center may be also specified in an argument to setimage using any valid direction. 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 qimager 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).

Finally, a position shift may be added using the arguments shiftx, shifty. This will be added to whatever the phase center was set to as described above. The shift is a real angle on the qimager so that, in e.g. RA, DEC, the RA shift is divided by cos(DEC) before being added to the RA. The sense of the shift is that the position after addition of the shift gives the new phase center of the image. The shift is in the reference frame of the phase center.

For spectral imaging setimage and setdata defines the spectral channels that are imaged. Examples are given in the setdata section.

Arguments

nx	Total number of spatial pixels in x Allowed: Int Default: 128
ny	Total number of spatial pixels in y Allowed: Int Default: 128
cellx	Cellsize in x (e.g. '1arcsec') Allowed: Quantity Default: '1arcsec'
celly	Cellsize in y (e.g. '1arcsec') Allowed: Quantity Default: '1arcsec'
stokes	Stokes parameters to image (e.g. 'IQUV') Allowed: 'I'|'IV'|'IQU'|'IQUV' Default: 'I'
doshift	Use the specified phase center? T or F Allowed: Bool Default: F
phasecenter	Direction of phase center as a measure Allowed: MDirection
shiftx	Shift in x (e.g. '23.7arcsec') Allowed: Quantity Default: '0arcsec'
shifty	Shift in y (e.g. '-54.2arcsec') Allowed: Quantity Default: '0arcsec'
mode	Type of processing Allowed: 'mfs'|'channel'|'velocity' Default: 'mfs'
nchan	Number of channels Allowed: Int Default: 1
start	Start channel (1-relative) Allowed: Int
step	Step in channel Allowed: Int
mstart	Start velocity Allowed: MRadialVelocity
mstep	Step in velocity Allowed: MRadialVelocity
spwid	Spectral Window Ids (1 relative) Allowed: Vector of Ints Default: 1
fieldid	Field Id (1 relative) Allowed: Int Default: 1
facets	Number of facets on each axis Allowed: Integer Default: 1
distance	Distance to object: usually ignore this! (m) Allowed: Quantity Default: '0m'

Example

```
## Example 1
imgr.setimage(nx=1024,ny=1024, cellx='30marcsec',celly='30marcsec',
nchan=1, stokes='IV', doshift=T, phasecenter=dm.direction('mars'));
## Example 2
imgr.setimage(nx=1024,ny=1024, cellx='30marcsec',celly='30marcsec',
nchan=1, stokes='IV', doshift=T,
phasecenter=image('othermarsimage').coordmeasures().direction);
## Example 3
myqimager.setdata(mode='channel', nchan=10, start=3, spwid=[1,2], fieldid=[3, 4, 5, 6, 7, 9, 10])
myqimager.setimage(nx=500, ny=500, mode='mfs', spwid=[1,2], fieldid=7)
myqimager.clean(algorithm='mfclark', niter=1000, model='mosaic')

## Example 4

dir1:=dm.direction('J2000', '20h00m00', '21d00m00')
dir2:=dm.direction('J2000', '20h10m00', '21d00m00')
dir3:=dm.direction('J2000', '20h00m00', '21d03m00')
imgr.setimage(nx=100, ny=100, cellx='0.1arcsec', celly='0.1arsec',
              doshift=T, phasecenter=dir1)
imgr.make('box1')
imgr.setimage(nx=100, ny=100, cellx='0.1arcsec', celly='0.1arsec',
              doshift=T, phasecenter=dir2)
imgr.make('box2')
imgr.setimage(nx=100, ny=100, cellx='0.1arcsec', celly='0.1arsec',
              doshift=T, phasecenter=dir3)
imgr.make('box3')
imgr.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 30marc, 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 that of the reference pixel of another image.
The third example shows the use of setdata and setimage to setup a mosaic. In the set data we have chosen 10 channels (for each IF) of data starting form channel 3. We also have selected IF 1 and 2. We have selected data from fields 3 to 10. In the setimage 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 images are made and deconvolved around known sources rather than making a big image englobing all three fields. Care should be taken in NOT overlapping the regions imaged this way otherwise the deconvolution will fail.

---

qimager.advise - Function

Package	synthesis
Module	qimager
Tool	qimager

Advise (and optionally use) parameter values

Synopsis
advise(takeadvice, amplitudeloss, fieldofview, pixels, cell, facets, phasecenter)

Description
Advise on recommended values of certain parameters. Return these values and optionally use them in Qimager.
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:

$$N_{facets} = N_{pixels} \sqrt{\frac{\frac{\Delta\theta}{\sqrt{8\delta A}} w_{rms}}{uv_{rms}}} \tag{1.24}$$

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

takeadvice	in	Use the advised values? Allowed: Bool Default: T
amplitudeloss	in	Maximum fractional amplitude loss due to faceting Allowed: Float Default: 0.05
fieldofview	in	Desired field of view Allowed: Quantity Default: '1deg'
pixels	out	Number of pixels on a side Allowed: Int
cell	out	Recommended maximum cellsize Allowed: Quantity
facets	out	Recommended number of facets on one axis Allowed: Integer
phasecenter	out	Direction of phase center as a measure Allowed: MDirection

---

qimager.make - Function

Package	synthesis
Module	qimager
Tool	qimager

Make an empty (i.e. blank) image

Synopsis
make(image, async)

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

image	name of output image AllowedString
async	Run asynchronously in the background AllowedBool Default!dowait

Returns
Bool

Example

```
imgr.setimage(nx=1024,ny=1024, cellx='30marcsec',celly='30marcsec',
nchan=1, stokes='IV', phasecenter=dm.direction('mars'));
imgr.make('mars.moving');
imgr.setimage(nx=1024,ny=1024, cellx='30marcsec',celly='30marcsec',
nchan=1, stokes='IV',
phasecenter=dm.direction('j2000', '12:23:48.7', '-15:56:32.9')
imgr.make('mars.fixed');
imgr.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.

---

qimager.regionmask - Function

Package	synthesis
Module	qimager
Tool	qimager

Construct a mask image from a region

Synopsis
regionmask(mask, region, value)

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 qimager 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. boxmask 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 'drm' can be used for most cases unless it is already set for some other images.
For gui users regions to be masked can be made from an image:

- Fill in the desired name of the mask.
- On the ''Spanner'' button for the region, select the 'FromImage'.
- This will pop a catalogue GUI, select the image you want to use a template to draw mask and press the 'send & Dismiss' button. This will display the image using the viewer.
- To set a polygonal region, click once on the bottommost button on the lefthand side of the viewer (clue: it looks like a polygon with a letter 'R' on it). Then set the vertices of the polygon by left-clicking each in turn. Finish with a double click on the last vertex and then another double click inside the polygon.
- In the region box a name of the region like 'myregion1' will appear.
- Press the ''Go'' button for qimager:regionmask.

Now to set multiple regions a 'union of regions' can be easily made by using the viewer in step 3 above. Under the Tools menu, select 'ImageAnalysis'. A box of rollups will appear. Select the region rollup. Select 'union' and press 'Start'. Then on the viewer click on one of the Region making buttons (the two bottom most buttons on the left margin with 'R's on them). After making a region click twice inside. The 'accumulate' on the Region rollup should increment. Keep repeating this process of making a region at different places and clicking twice in (remember 'ESC' key is used to get rid of a box you have drawn before drawing the next one). Press on the 'Finish' button and dismiss the rollup and viewer. A region with a name like 'myregion1' would have appeared in the region box, which represent a union of all the regions the user has selected. For script writers who still want to interactively set the regions, a call to the gopherwidgets:gopher tool can used.

Arguments

mask	name of mask image Allowed String
region	Region Allowed Any valid region or a record of regions Default unset
value	Value to set the mask to Allowed Any scalar Default 1.0

Returns
Bool

Example

The following the image bigmask, and then sets the inner quarter to unity. Then cleans using it as the mask.

```
- imgr.regionmask(mask='bigmask', region=drm.quarter())
- imgr.clean(mask='bigmask', model='3C273XC1.clean.masked', niter=1000)
```

Another example using drm.wbox function:

```
- im:=image('dirty')
- cs:=im.coordsys()
```

```
- drm.setcoordinates(cs,F)
- r1:=drm.wbox(blc=['173pix', '347pix'], trc=['183pix', '370pix'])
- imagr.regionmask(mask='bigmask',region=r1)
```

Or using a record of regions:

```
- r2:=drm.wbox(blc=['180pix', '344pix'], trc=['191pix', '369pix'])
- r3:=drm.wbox(blc=['189pix', '341pix'], trc=['204pix', '364pix'])
- rec:=[=]
- rec[1]:=r1
- rec[2]:=r2
- rec[3]:=r3
- imagr.regionmask(mask='bigmask',region=rec)
```

Now one may wish to make regions and later use it with higher resolution images etc. Regions can be stored in tables on disk and recovered later for re-use (especially if its a long series of regions making using the gui as described above.

```
- drm.fromglobaltotable('mytab.region', 'T', 'F', "myregionoftoday", myregion1)
```

The above command using the default regionmanager tool saves the region 'myregion1' into a table 'mytab.region' Subsequently this table is used to recover the region and use it to create a mask image as below. We use the function drm.namesintable() to remind ourselves under what names the regions was saved and hence allow us to pick the one we want.

```
- drm.namesintable('mytab.region')

myregionoftoday

- myoldregion:=drm.fromtabletoglobal('mytab.region', T, 'myregionoftoday')

- imagr.regionmask(mask='newmask',region=myoldregion)
```

---

qimager.exprmask - Function

Package	synthesis
Module	qimager
Tool	qimager

Construct a mask image from a LEL expression

Synopsis
exprmask(mask, expr)

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 qimager 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

mask	name of mask image Allowed String
expr	Value to set the mask to Allowed Any scalar or LEL expression Default 1.0

Returns
Bool

Example

```
imgr.exprmask(mask='bigmask', expr='"3C273XC1.clean">0.5')
imgr.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.

---

qimager.makeimage - Function

Package synthesis
Module qimager
Tool qimager

Calculate images by gridding, etc.

Synopsis
makeimage(type, image, compleximage, async)

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

model Make the dirty image from the MODEL_DATA column

corrected Make the dirty image from the CORRECTED_DATA column (*default*)

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

Note the full qimager 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 qimager equation where primary beam correction is performed.
A position shift can be applied when specifying the image parameters with setimage. 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:

```
include 'image.g';
imcov  := image('scanweight'); imcov.view();
s:=0; imcov.statistics(s);
threshold := s.max / 10.0;
#
im:=imagecalc('sdimage',
      pixels=spaste('scanimage[scanweight>', threshold,
    ']/scanweight[scanweight>', threshold, ']'))
im.view(raster=T, axislabels=T);
```

Arguments

type	Type of output image Allowed String
image	Name of output image Allowed String
compleximage	Name of output complex image Allowed String
async	Run asynchronously in the background Allowed Bool Default !dowait

Returns
Bool

Example

```
imgr.ft(model='3C273XC1.model', complist='3C273XC1.complist');
imgr.image(type='residual', image='3C273XC1.residual')
imgr.image(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.

---

qimager.weight - Function

Package	synthesis
Module	qimager
Tool	qimager

Apply additional weighting to the visibility weights

Synopsis
weight(type, rmode, noise, robust, fieldofview, npixels, async)

Description
Apply visibility weighting to correct for the local density of sampling in the uv plane. The imaging weights are written into a Table column called IMAGING_WEIGHT, which may be plotted using plotweights. In addition this columns may be accessed directly using either the table or ms modules.
To correct for visibility sampling effects, natural, uniform (the default), 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:

$$w_i = \frac{1}{\sigma_i^2} \tag{1.25}$$

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:

$$w_i = w_i \sqrt{u_i^2 + v_i^2} \quad (1.26)$$

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:

$$w_i = \frac{w_i}{W_k} \quad (1.27)$$

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:

$$w_i = \frac{w_i}{1 + W_k f^2} \quad (1.28)$$

where:

$$f^2 = \frac{(5 * 10^{-R})^2}{\frac{\sum_k W_k^2}{\sum_i w_i}} \quad (1.29)$$

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:

$$w_i = \frac{w_i}{W_k * R^2 + 2 * \sigma_R^2} \quad (1.30)$$

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 sigma (expected noise) per visibility (actually stored in the SIGMA column).

Arguments

type	Type of weighting Allowed: String: 'uniform'|'natural'|'briggs'|'radial' Default: uniform
rmode	Mode of briggs weighting Allowed: String: 'norm'|'abs'|'none' Default: none
noise	Noise used in absolute briggs weighting Allowed: Quantity Default: '0.0Jy'
robust	Parameter in briggs weighting Allowed: Double: range -2.0 to 2.0 Default: 0.0
fieldofview	Field of view for uniform weighting Allowed: Quantity Default: '0arcsec'
npixels	Number of pixels in the u and v directions Allowed: Int Default: 0
async	Run asynchronously in the background Allowed: Bool Default: !dowait

Returns
Bool

Example

```
imgr.weight(type='briggs', rmode='norm', robust=0.5)
```

Applies Briggs (robust) weighting.

---

qimager.sensitivity - Function

Package	synthesis
Module	qimager
Tool	qimager

Calculate rms sensitivity

Synopsis
sensitivity(pointsource, relative, sumweights, async)

Description
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) and the specified sigma per point (in the column SIGMA). Therefore, this includes all the effects of weight and filter.

Arguments

pointsource	out	Calculated point source sensitivity (Jy/beam) Allowed:Quantity
relative	out	Calculated relative sensitivity Allowed:Double
sumweights	out	Calculated sum of weights Allowed:Double
async	in	Run asynchronously in the background Allowed:Bool Default:!dowait

Returns
Bool

Example

```
local s,r,sum
ok := imgr.sensitivity(s,r,sum,F)
print 'Sensitivity = ', s
print 'Relative to Natural Weighting = ', r
```

Example

```
imgr.sensitivity()
# Wait for it to finish and then ask for the result:
- defaultservers.result(1)
[pointsource=[value=0.25, unit='Jy'], relative=2.5, sumweights=642085.45]
```

---

qimager.filter - Function

Package synthesis
Module qimager
Tool qimager

Apply additional weighting by filtering (u-v taper)

Synopsis
filter(type, bmaj, bmin, bpa, async)

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 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 qimager.fitpsf.

Arguments

type	Type of filtering or u-v tapering Allowed String Default gaussian
bmaj	Major axis of filter Allowed Quantity Default '0rad'
bmin	Minor axis of filter Allowed Quantiy Default '0rad'
bpa	Position angle of filter Allowed Quantity Default '0deg'
async	Run asynchronously in the background Allowed Bool Default !dowait

Returns
Bool

Example

```
imgr.weight('uniform')
imgr.filter(type='gaussian', bmaj='2.3arcsec', bmin='1.67arcsec',
bpa='-34.5deg')
```

---

qimager.uvrange - Function

Package	synthesis
Module	qimager
Tool	qimager

Select data within the limit of a given range

Synopsis
uvrange(uvmin, uvmax)

Description
Apply a uvrange so that only points within a given uvrange are selected for further usage. To be noted setdata if used after

uvrange will reset the selected range. So setdata 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

uvmin	Minimum uv distance allowed (wavelengths) Allowed Float Default 0.0
uvmax	Maximum uv distance allowed (wavelengths) Allowed Float Default 0.0

Returns
Bool

Example

```
imgr.weight('uniform')
imgr.uvrange(0, 4000.0)
```

---

qimager.fitpsf - Function

Package	synthesis
Module	qimager
Tool	qimager

Fit the point spread function, making psf image first if needed

Synopsis
fitpsf(psf, bmaj, bmin, bpa, async)

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 setdata, setimage or any tool function that changes the weights. Use the function summary to check if there is a valid fitted psf stored internally.

Arguments

psf	Name of input psf Allowed String
bmaj	Major axis of beam Allowed Quantity record
bmin	Minor axis of beam Allowed Quantity record
bpa	Position angle of beam Allowed Quantity record
async	Run asynchronously in the background Allowed Bool Default !dowait

Returns
Bool

Example

```
- imgr.makeimage(type='psf', image='3C273XC1.psf')
- majaxis:=[=]; minaxis:=[=]; pangle:=[=]
- imgr.fitpsf('3C273XC1.psf', bmaj=majaxis, bmin=minaxis, bpa=pangle, async=F)
- print bmaj, bmin, bpa
- imgr.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 following example:

```
- imgr.fitpsf(psf='')
- imgr.restore(model='bla' , complist='', image='bla.restored' , residual='bla2.residual' )
```

---

qimager.setbeam - Function

Package	synthesis
Module	qimager
Tool	qimager

Set the beam parameters for clean restoration

Synopsis
setbeam(bmaj, bmin, bpa, async)

Description
This sets the clean beam that will be used in all restoration operations.

Arguments

bmaj	Major axis of beam Allowed Quantity record
bmin	Minor axis of beam Allowed Quantity record
bpa	Position angle of beam Allowed Quantity record
async	Run asynchronously in the background Allowed Bool Default !dowait

Returns
Bool

---

qimager.clean - Function

Package	synthesis
Module	qimager
Tool	qimager

Calculate a deconvolved image with selected clean algorithm

Synopsis
clean(algorithm, niter, gain, threshold, displayprogress, model, fixed, complist, mask, image, residual, interactive, npercycle, masktemplate, async)

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).
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 setimage 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 setdata and setimage. Before clean can be run with a multi-field algorithm, 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=T 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. Use 'refresh mask' to review the recent changes in the mask. 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. PLEASE be patient and wait for the viewer to load both the image and mask before clicking on the buttons or drawing regions. If impatient please use interactive=F. Even if interactive=F, 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 a prior image to make the mask the first time (e.g a previously cleaned image)

Arguments

algorithm	Algorithm to use Allowed:String: 'clark'|'hogbom'|'multiscale'|'mfclark'|'csclean'|'csfast'|'mfhogbom'|'mfmultiscale'|'wfclark'|'wfhogbom' Default:'clark'
niter	Number of Iterations, set to zero for no CLEANing Allowed:Int Default:1000
gain	Loop Gain for CLEANing Allowed:Float Default:0.1
threshold	Flux level at which to stop CLEANing Allowed:Quantity Default:'0Jy'
displayprogress	Display the progress of the cleaning? Allowed:Bool Default:F
model	Names of clean model images Allowed:Vector of strings
fixed	Keep one or more models fixed Allowed:Vector of booleans Default:F
complist	Name of component list Allowed:String
mask	Names of mask images used for CLEANing Allowed:Vector of strings
image	Names of restored images Allowed:Vector of strings
residual	Names of residual images Allowed:Vector of strings
interactive	whether to stop clean and interactively mask Allowed:Boolean Default:F
npercycle	If interactive is 'T', then no of iter of clean before stopping, usually a fraction of niter Allowed:Int Default:100
masktemplate	If non empty then will use this image to make the mask the first time Allowed:String Default:''
async	Run asynchronously in the background? Allowed:Bool Default:!dowait

Returns
Bool

Example

```
imgr.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:

- 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.
- 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.
- No restored image is made if the image string is explicitly unset.

```
include 'qimager.g';
msfile := 'vlac125K.ms';
imgr:=qimager(msfile);
npix := 500; cell:='5arcsec';
#
# CS on 500 by 500
#
imgr.setimage(nx=npix, ny=npix, cellx=cell, celly=cell, stokes='I',
      spwid=[1,2]);i
imgr.setoptions(padding=1.0);
imgr.setdata(spwid=[1,2]);
imgr.clean('cs', model='vlac125K.cs', image='vlac125K.cs.restored',
   niter=1000, gain=0.1);
#
# CSF on 1000 by 1000, cleaning inner quarter
#
include 'regionmanager.g';
imgr.setimage(nx=2*npix, ny=2*npix, cellx=cell, celly=cell, stokes='I',
      spwid=[1,2]);
imgr.regionmask('vlac125K.mask', region=drm.quarter());
imgr.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
#
include 'regionmanager.g';
imgr.setimage(nx=2*npix, ny=2*npix, cellx=cell, celly=cell, stokes='I',
      spwid=[1,2]);
imgr.clean('cs', model='vlac125K.csl', image='vlac125K.csl.restored',
   mask='vlac125K.mask', niter=1000, gain=0.1);

imgr.done();
```

---

qimager.mem - Function

Package	synthesis
Module	qimager
Tool	qimager

Calculate a deconvolved image with selected mem (maximum entropy) algorithm

Synopsis
mem(algorithm, niter, sigma, gain, targetflux, constrainflux, displayprogress, model, fixed, complist, prior, mask, image, residual, async)

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.
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 setdata and setimage. 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

algorithm	Algorithm to use Allowed String:'entropy'|'emptiness'|'mfentropy'|'mfemptiness' Default:'entropy'
niter	Number of Iterations Allowed Int Default 20
sigma	Image sigma to try to achieve Allowed Quantity Default:'0.001Jy'
gain	Gain for step Allowed Float Default 0.3
targetflux	Target flux for final image Allowed Quantity Default:'1.0Jy'
constrainflux	Constrain image to match target flux? else targetflux used only to initialize model Allowed Bool Default F
displayprogress	Display the progress of the cleaning? Allowed Bool Default F
model	Names of model images Allowed Vector of strings
fixed	Keep model fixed Allowed Vector of booleans Default F
complist	Name of component list Allowed String
prior	Names of mem prior images Allowed Vector of strings
mask	Names of mask images (0=>no emission, 1=>emission permitted Allowed Vector of strings
image	Names of restored images Allowed Vector of strings
residual	Names of residual images Allowed Vector of strings
async	Run asynchronously in the background? Allowed Bool Default:!dowait

Returns
Bool

Example

```
imgr.mem(model='3C273XC1.mem.model',
mask='3C283XC1.mask', niter=40, sigma='0.001Jy')
```

---

qimager.nnls - Function

Package	synthesis
Module	qimager
Tool	qimager

Calculate a deconvolved image using the NNLS algorithm

Synopsis

```
nnls(model, fixed, complist, niter, tolerance, fluxmask,
datamask, image, residual, async)
```

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 AIPS++ 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

model	Name of image Allowed: Vector of strings
fixed	Keep model fixed Allowed: Vector of booleans Default: F
complist	Name of component list Allowed: String
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: Vector of strings
datamask	Name of mask for constraint pixels in dirty image Allowed: Vector of strings
image	Names of restored images Allowed: Vector of strings
residual	Names of restored images Allowed: Vector of strings
async	Run asynchronously in the background Allowed: Bool Default: !dowait

Returns
Bool

Example

```
imgr.nnls(image='3C273XC1.nnls.image', model='3C273XC1.nnls.model',
fluxmask='3C283XC1.fluxmask', datamask='3C273XC1.datamask', niter=1000,
tolerance=0.00001)
```

qimager.setmfcontrol - Function

Package	synthesis
Module	qimager
Tool	qimager

Set various cycle control parameters for multi-field and wide-field imageing.

Synopsis
setmfcontrol(cyclefactor, cyclespeedup, stoplargenegatives, stoppointmode)

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.

Arguments

cyclefactor	Cycle threshold = this * max sidelobe * max resid Allowed: Float Default: 1.5
cyclespeedup	Cycle threshold doubles in this number of iterations Allowed: Float Default: -1
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

Returns
Bool

Example

```
imgr.setmfcontrol(cyclefactor=2.0, cyclespeedup=niter/10,
stoplargenegatives=T, stoppointmode=10);
```

---

qimager.feather - Function

Package	synthesis
Module	qimager
Tool	qimager

Feather together an interferometer and a single dish image in the Fourier plane

Synopsis
feather(image, highres, lowres, lowpsf, usedefaultvp, vptable, async)

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 approximatepsf.

Arguments

image	Name of output feathered image Allowed: Image Default: 'feathered.image'
highres	Name of high resolution (interferometer) image Allowed: Image
lowres	Name of low resolution (single dish) image Allowed: Image
lowpsf	Name of optional low resolution point spread function Allowed: Image
usedefaultvp	Use the default vp type? Allowed: Bool Default: T
vptable	Voltage pattern table from the vpmanager for detailed specification Allowed: Table
async	Run asynchronously in the background Allowed: Bool Default: !dowait

Returns
Bool

Example

```
imgr.feather(image='feathered.image', highres='casa.vlaonly',
lowres='casa.sd', usedefaultvp=F, vptable='sd.vptable');
```

---

qimager.pb - Function

Package	synthesis
Module	qimager
Tool	qimager

Applies or corrects for a primary beam

Synopsis
pb(inimage, outimage, incomps, outcomps, operation,
pointingcenter, parangle, pborvp, async)

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==F 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 T.

The applied primary beam function is deterimed as follows. If you used function Qimager.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

inimage	Input image to apply beam to Allowed String Default None
outimage	Output image after beam is applied Allowed String Default No output image
incomps	Input Componentlist table name Allowed String Default None
outcomps	Output Componentlist table name Allowed String Default No output Componentlist
operation	Operation Allowed String from 'apply' or 'correct' Default 'apply'
pointingcenter	Pointing center for primary beam application Allowed F or Direction measure Default F in CLI or the N. pole in the GUI
parangle	Parallactic angle for calculation Allowed Quantity Default '0deg'
pborvp	Primary Beam or Voltage Pattern Allowed String from 'pb' or 'vp' Default 'pb'
async	Run asynchronously in the background Allowed Bool Default !dowait

Returns
Bool

Example

```
# make a flat image
imgr.make('flat.image');
img := image('flat.image');
arr := img.getchunk();
arr[1:shape(arr)[1], 1:shape(arr)[2]] := 1.0;
img.putchunk(arr);
img.done()
arr := F;
#
# as we are using "pointingcenter=F", it defaults to the image center
imgr.pb(inimage='flat.image', outimage='pb.image', pointingcenter=F)
```

qimager.linearmosaic - Function

Package	synthesis
Module	qimager
Tool	qimager

Make a linear mosaic of several images

Synopsis
linearmosaic(images, mosaic, fluxscale, sensitivity, fieldids, usedefaultvp, vptable, async)

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

images	Input images to be mosaiced Allowed: Vector of strings
mosaic	Output mosaic image Allowed: Image
fluxscale	Fluxscale image Allowed: Image
sensitivity	Sensitivity image Allowed: Image
fieldids	output component list after PB has been applied [optional] Allowed: Vector of Ints
usedefaultvp	Use the default vp type? Allowed: Bool Default: T
vptable	Voltage pattern table from the vpmanager for detailed specification Allowed: Table
async	Run asynchronously in the background Allowed: Bool Default: !dowait

Returns
Bool

Example

```
imgr.linearmosaic(images=['orion.1.cln', 'orion.2.cln', 'orion.4.cln'], mosaic='orion.linmos',
fluxscale='orion.linmos.fluxscale', fieldid=[1,2,4]);
```

---

qimager.predict - Function

Package	synthesis
Module	qimager
Tool	qimager

Predict the data from the specified model and componentlist

Synopsis
predict(model, complist, incremental, async)

Description
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

model	Name of image Allowed:Vector of strings
complist	Name of component list Allowed:String
incremental	Add to the existing MODEL_DATA column? Allowed:Bool Default:F
async	Run asynchronously in the background? Allowed:Bool Default:!dowait

Returns
Bool

Example

```
imgr.ft(model='3C273XC1.nnls.model')
imgr.ft(model='3C273XC1.another.model', incremental=T)
```

Fourier transforms the model in the image 3C273XC1.nnls.model and then adds the visibility due to 3C273XC1.another.model

---

qimager.approximatepsf - Function

Package	synthesis
Module	qimager
Tool	qimager

Calculate approximate point spread functions

Synopsis
approximatepsf(model, psf, async)

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

model	Names of input models Allowed: Vector of Strings	
psf	Names of output point spread functions Allowed: Vector of Strings	
async	Run asynchronously in the background Allowed: Bool Default: !dowait	

Returns
Bool

Example

```
- imgr.approximate(model='3C273XC1.clean', psf='3C273XC1.clean.psf')
```

---

qimager.restore - Function

Package	synthesis
Module	qimager
Tool	qimager

Calculate the restored image with restored model, component list, and residuals

Synopsis
restore(model, complist, image, residual, async)

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

model	Names of input model Allowed:Vector of Strings
complist	Name of component list Allowed:String
image	Names of output restored images Allowed:Vector of Strings
residual	Names of residual images Allowed:Vector of strings
async	Run asynchronously in the background Allowed:Bool Default:!dowait

Returns
Bool

Example

```
- imgr.setbeam(bmaj='2.0arcsec', bmin='2.0arcsec')
- imgr.restore(model='3C273XC1.clean', image='3C273XC1.clean.restored',
```

---

qimager.residual - Function

Package	synthesis
Module	qimager
Tool	qimager

Calculate the residual image with respect to current model and component list

Synopsis
residual(model, complist, image, async)

Description
Calculate the residuals corresponding to the model and componentlist. *Note that the model visibilities are updated.*

Arguments

model	Names of input models AllowedVector of Strings
complist	Name of component list AllowedString
image	Names of output residual images AllowedVector of Strings
async	Run asynchronously in the background AllowedBool Default!dowait

Returns
Bool

Example

```
- imgr.residual(model='3C273XC1.clean', complist='3C273XC1.cl',
image='3C273XC1.clean.residual')
```

---

qimager.smooth - Function

Package	synthesis
Module	qimager
Tool	qimager

Calculate an image smoothed with a Gaussian beam

Synopsis
smooth(model, image, usefit, bmaj, bmin, bpa, normalize, async)

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

model	Name of input model Allowed: Vector of Strings
image	Name of output smoothed images Allowed: Vector of Strings
usefit	Use the fitted value (rather than that specified Allowed: Boolean Default: T
bmaj	Major axis of beam Allowed: Quantity record
bmin	Minor axis of beam Allowed: Quantity record
bpa	Position angle of beam Allowed: Quantity record
normalize	Normalize volume of psf to unity Allowed: Bool Default: T
async	Run asynchronously in the background Allowed: Bool Default: !dowait

Returns
Bool

Example

```
- imgr.smooth(model='3C273XC1.clean', image='3C273XC1.clean.restored',
bmaj='2.0arcsec', bmin='2.0arcsec')
```

---

qimager.setoptions - Function

Package	synthesis
Module	qimager
Tool	qimager

Set some general options for subsequent processing

Synopsis
setoptions(ftmachine, cache, tile, gridfunction, padding)

Description
Various less-often-used options for processing can be set.

ftmachine The options for ftmachine are:

- **wproject** W projection algorithm.
- **mosaic** Mosaic 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. are supported.

Arguments

ftmachine	Fourier transform machine Allowed:String:'ft'|'sd'|'both' | 'wfmemoryft' Default:'ft'
cache	Size of gridding cache in complex pixels Allowed:Int Default:4194304
tile	Size of a gridding tile in pixels (in 1 dimension) Allowed:Int Default:16
gridfunction	Gridding function Allowed:String: 'SF'|'BOX' Default:'SF'
padding	Padding factor in image plane (>=1.0) Allowed:Float Default:1.0

Returns
Bool

Example

```
- imgr.setoptions(cache=10000000, tile=32, gridfunction='BOX',
  dm.location('vla'))
```

---

qimager.setsdoptions - Function

Package	synthesis
Module	qimager
Tool	qimager

Set some options for single dish processing

Synopsis
setsdoptions(scale, weight)

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.

cache The size of the cache used (in complex pixels) during the

Arguments

scale		Scaling applied to single dish data
	Allowed	Float
	Default	1.0
weight		Weights applied to single dish data
	Allowed	Float
	Default	1.0

Returns
Bool

Example

```
- imgr.setoptions(cache=10000000, tile=32, gridfunction='BOX',
  dm.location('vla'))
```

---

qimager.setvp - Function

Package synthesis
Module qimager
Tool qimager

Set the voltage pattern model for subsequent processing

Synopsis

setvp(dovp, usedefaultvp, vptable, dosquint, parangleinc)

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 apprорpiate 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

dovp	Do voltage pattern (ie, primary beam) correction Allowed Bool Default F
usedefaultvp	Look up the default VP for this telescope and frequency? Allowed Bool Default T
vptable	If usedefaultvp is false, provide a VP Table made with vpmanager Allowed String
dosquint	Activate the beam squint in the VP model Allowed Bool Default F
parangleinc	Parallactice angle increment for squint application Allowed Quantity Default 360deg

Returns
Bool

Example

```
- imgr.setvp(dovp=T, usedefaultvp=T, dosquint=F);
```

---

qimager.setscales - Function

Package	synthesis
Module	qimager
Tool	qimager

Set the scale sizes for MultiScale Clean

Synopsis
setscales(scalemethod, nscales, uservector)

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:

$$\theta_{minor} 10.0^{(i-N_{scales}/2)/2.0} \tag{1.31}$$

where θ_{min} is the fitted minor axis of the clean beam. The first value is zero.

Arguments

scalemethod	Method by which scales are set Allowed String: 'nscales'|'uservector' Default nscales
nscales	Number of scales Allowed Int Default 5
uservector	Vector of scale sizes to use Allowed Vector of Doubles Default [0, 3, 10]

Returns
Bool

Example

```
- imgr.setscales(6);
```

---

qimager.selfcal - Function

Package	synthesis
Module	qimager
Tool	qimager

Self-calibrate the MeasurementSet using a calibrater tool and model+component list

Synopsis
selfcal(caltool, model, complist)
To self-calibrate, first make a calibrater tool and set it up as required. Then use this tool function to do the selfcal. The steps coded in the tool function may also be done by hand, of course. These are:

1. Call qimager.ft To Fourier transform the model and optional componentlist
2. Call calibrater.solve to do the solution, optionally writing out tables containing gain information.
3. Call calibrater.correct to do the correction of the data.

After the selfcal step, the next stage is to re-image the data using *e.g.* clean or nnls.

Description

Arguments

caltool	Glish name of calibrater tool Allowed: Any valid calibrater tool Default: NONE!
model	Name of images Allowed: Vector of strings
complist	Name of component list Allowed: String

Returns
Bool

Example

```
imgr:=qimager('3C273XC1.ms')
cl:=calibrater('3C273XC1.ms')
cl.setsolve('T', 60, 'T', -1, '', F)
imgr.selfcal(cl, model='clean')
imgr.clean(model='newclean', image='newclean.restored')
```

1.5.2 qimagermaketestms - Function

Package synthesis
Module qimager

Make a standard MeasurementSet that can be used for experimentation

Synopsis
qimagermaketestms(msfile)

Description
A standard dataset is converted from a FITS file to a MeasurementSet. The dataset used is an observation of 3C273 at 8GHz using the C-configuration of the VLA. It is part of the AIPS++ distribution.

Returns
Bool

Arguments

msfile	Name of output ms	
	Allowed	String
	Default:	'3C273XC1.ms'

Example

```
qimagermaketestms('MY3C273.ms')
```

1.5.3 qimagermaketestsdms - Function

Package synthesis
Module qimager

Make a standard single dish MeasurementSet that can be used for experimentation

Synopsis
qimagermaketestsdms(msfile)

Description
A standard dataset is converted from a set of GBT fits files to a MeasurementSet. The dataset used is an observation of the Cygnus loop at 800MHz using GBT. It is part of the AIPS++ distribution.

Returns
Bool

Arguments

msfile	Name of output ms	
	Allowed	String
	Default:	'gbt_cygnus_800MHz.ms'

Example

```
qimagermaketestsdms('mySD.ms')
```

1.5.4 qimagermaketestmfms - Function

Package synthesis
Module qimager

Make a standard multifield MeasurementSet that can be used for experimentation

Synopsis
qimagermaketestmfms(msfile)

Description
A standard multifield dataset is converted from a FITS file to a MeasurementSet. The dataset used is a seven field VLA D-array observation of CAS A at 8 GHz and is part of the AIPS++ distribution.

Returns
Bool

Arguments

msfile	Name of output ms Allowed String Default:'XCAS.ms'	

Example

```
qimagermaketestmfms('XCAS.ms')
```

1.5.5 qimagermaketestcl - Function

Package synthesis
Module qimager

Make a standard componentlist for a standard MeasurementSet

Synopsis
qimagermaketestcl(clfile, refer)

Description
A standard component list is made for the 3C273XC1 data set. The reference for the component can be specified: the components will be then converted to this reference (e.g. 'gal') before processing.

Returns
Bool

Arguments

clfile	Name of output cl Allowed String Default '3C273XC1.cl'
refer	Reference Allowed String Default 'gal'

Example

```
qimagermaketestcl('MY3C273.cl', 'gal')
```

1.5.6 qimagertest - Function

Package synthesis
Module qimager

Perform a test of imaging from a standard MeasurementSet

Synopsis
qimagertest(size, cleanniter, cleangain, doshift, doplot)

Description
Perform a test. The dataset used is an observation of 3C273 at 8GHz using the C-configuration of the VLA. It is part of the AIPS++ distribution.
Note that if doshift is set to T then the imaging is done in Galactic coordinates.
The test does the following:

- The uvfits file that is to be read in is in a standard place in the AIPS++ code source area.
- The MeasurementSet is read from the uvfits file.
- A summary of what exists is logged.
- The data are weighted using the Briggs algorithm.
- An initial CLEAN-only deconvolution is performed, and then restored.
- A CLEAN mask is constructed by thresholding $I > 0.015$

- The deconvolution is repeated with masks enabled.
- Residual images are calculated for the North Celestial pole, on the Sun, and for azimuth elevation (0d, 0d).

Returns
Bool

Arguments

size	Number of pixels on a side Allowed: Int Default: 128
cleanniter	Number of clean iterations Allowed: Int Default: 1000
cleangain	Clean loop gain Allowed: Float Default: 0.1
doshift	Shift the phase center Allowed: Bool Default: T
doplot	Plot? Allowed: Bool Default: T

Example

```
include 'qimager.g'
qimagertest(size=256)
```

1.5.7 qimagerlongtest - Function

Package synthesis
Module qimager

Perform a long test of imaging from a standard MeasurementSet

Synopsis
qimagerlongtest()

Description
Perform a test. The dataset used is an observation of 3C273 at 8GHz using the C-configuration of the VLA. It is part of the AIPS++ distribution.
The test is similar to qimagertest, except that the various functions are checked one by one and status reported.

Example

```
include 'qimager.g'
qimagerlongtest()
```

1.5.8 qimagermftest - Function

Package synthesis
Module qimager

Perform a test of multi-field processing using a standard MeasurementSet

Synopsis
qimagermftest(doplot)

Description
Perform a test of multi-field processing. The dataset used is an observation of 3C273 at 8GHz using the C-configuration of the VLA. It is part of the AIPS++ distribution.
Note that if doshift is set to T then the imaging is done in Galactic coordinates.
The test does the following:

- The uvfits file that is to be read in is in a standard place in the AIPS++ code source area.
- The MeasurementSet is read from the uvfits file.
- The data are weighted using the Briggs algorithm.
- Two fields are cleaned, one on the core and one on the jet, using the standard component list.
- If doplot=T then the visibilities are plotted.

Returns
Bool

Arguments

doplot	Plot?	
	Allowed	Bool
	Default	T

Example

```
qimagermftest()
```

1.5.9 qimagerspectraltest - Function

Package synthesis
Module qimager

Perform a test of spectral processing using a standard spectral line MeasurementSet

Synopsis
qimagerspectraltest()

Description
Perform a test. The dataset used is an observation of BLLAC at 110 GHz using the BIMA array. It is part of the AIPS++ distribution.
The test does the following:

- The uvfits file that is to be read in is in a standard place in the AIPS++ code source area.
- The MeasurementSet is read from the uvfits file.
- An CLEAN-only deconvolution is performed, and then restored.

Returns
Bool

Example

```
qimagerspectraltest()
```

1.5.10 qimagerpbtest - Function

Package	synthesis
Module	qimager

Perform a test of primary beam processing using a standard MeasurementSet

Synopsis
qimagerpbtest()

Description
Perform a test of primary processing. The dataset used is an observation of 3C273 at 8GHz using the C-configuration of the VLA. It is part of the AIPS++ distribution.
The test does the following:

- The uvfits file that is to be read in is in a standard place in the AIPS++ code source area.
- The MeasurementSet is read from the uvfits file.
- A default calibration is applied
- The data are weighted using the Briggs algorithm.
- One field is cleaned, using the mf algorithm and the primary beam is applied both during the subtraction of components (the major cycle) and in estimating the residuals.

Returns
Bool

Example

```
qimagerpbtest()
```

1.5.11 qimagercomponenttest - Function

Package synthesis
Module qimager

Perform a test of component model processing using a standard MeasurementSet

Synopsis
qimagercomponenttest(size, doshift, doplot)

Description
Perform a test of component model processing. The dataset used is an observation of 3C273 at 8GHz using the C-configuration of the VLA. It is part of the AIPS++ distribution.
The test does the following:

- The uvfits file that is to be read in is in a standard place in the AIPS++ code source area.
- The MeasurementSet is read from the uvfits file.
- A component list is constructed and using in the subsequent processing. A point component is placed at the position of the core of 3C273, but specified in Galactic coordinates.
- The component model is used in predicting the model visibilities.
- A plot of the visibilities is made.
- The image is cleaned and restored. The center of the image is shifted to half way down the jet.

Returns
Bool

Arguments

size	Number of pixels on a side Allowed:Int Default:128
doshift	Shift the phase center Allowed:Bool Default:T
doplot	Plot? Allowed:Bool Default:T

Example

```
qimagercomponenttest()
```

1.5.12 qimagerselfcaltest - Function

Package synthesis
Module qimager

Perform a test of selfcal processing using a standard MeasurementSet

Synopsis
qimagerselfcaltest(size, doshift, doplot)

Description
Perform a test of selfcal model processing. The dataset used is an observation of 3C273 at 8GHz using the C-configuration of the VLA. It is part of the AIPS++ distribution.
The test does the following:

- The uvfits file that is to be read in is in a standard place in the AIPS++ code source area.
- The MeasurementSet is read from the uvfits file.
- qimager and calibrater tools are created.
- qimager is used to clean and restore the image
- The clean model is clipped at 750mJy

- The selfcal tool function is used to self-calibrate using the current model. The Jones matrices are: T, 60s and G, 600s.
- The clean and restoration are repeated with the improved calibration.

Returns
Bool

Arguments

size	Number of pixels on a side Allowed: Int Default: 128
doshift	Shift the phase center Allowed: Bool Default: T
doplot	Plot? Allowed: Bool Default: T

Example

```
qimagerselfcaltest()
```

1.5.13 qimagersdtest - Function

Package synthesis
Module qimager

Perform a test of single dish imaging from a standard MeasurementSet

Synopsis
qimagersdtest()

Description
Perform a test. The dataset used is an observation of the Cygnus

region at 800 MHz using the GBT. It is part of the AIPS++ distribution.
The test does the following:

- The GBT FITS files that are to be read in are in a standard place in the AIPS++ data respository.
- The MeasurementSet is created using the gbtmsfiller
- The data are calibrated using gbtcal.
- Two images are constructed: the data gridded and the weights gridded.
- The image calculator is used to normalize by the weights image.

Returns
Bool

Example

```
include 'qimager.g'
qimagersdtest();
```

1.5.14 qimageralltests - Function

Package synthesis
Module qimager

Perform all qimager tests using standard MeasurementSets

Synopsis
qimageralltests()

Description
All tests are run and the status returned. The complete suite takes about 500s (real-time) on a Pentium Pro 200MHz.

1.5.15 qimagermultiscale - Function

Package synthesis
Module qimager

Makes multi-resolution multiscale images.

Synopsis
qimagermultiscale(msname, imsizes, cellsizes, scales, nitermult, niterpower, fields, spwid, centerfield)

Description
Based on user-specified image sizes and cell sizes, we make several rounds of multiscale clean images and use each model image as a fisr guess for successively higher resolution images. After the first image, each clean is solving for deltas, and will tend to be neither positive nor negative.

For each round of multiscale cleaning, uniform weighting with a uv taper coresponding to 2.5 times the cell size is used. The number of multiscale clean iterations used in each round is given by

$$niter = nitermult * imsizes[i]^{niterpower}. \qquad (1.32)$$

Returns
Bool

Arguments

msname	Name of input ms Allowed:String Default:''
imsizes	Image sizes for each round (should be monatonically increasing) Allowed:Vector of Ints Default:[128, 256]
cellsizes	Cell sizes [in arcsec] for each round (should be monatonically decreasing) Allowed:Vector of Floats Default:[2, 1]
scales	Name of input ms Allowed:Vector of Ints Default:[0, 5, 15]
nitermult	Multiplier parameter used to adjust niter with imsizes Allowed:Float Default:0.15
niterpower	Exponent of imsizes used to adjust niter with imsizes Allowed:Float Default:1.2
fields	Vector of field ids to image Allowed:Vector of Ints Default:[1]
spwid	Spectral window to image Allowed:Int Default:1
centerfield	Field id for central pointing Allowed:Int Default:1

Example

```
qimagermultiscale(msname='orion.ms', imsizes=[128, 256, 512], cellsizes=[16,8,4],
scales=[0, 5, 15], nitermult=0.15, niterpower=1.2, fields=[1:4],
spwid=[1], centerfield=1);
```