AIPS++ User Reference Manual - BIMA Package

AIPS++ Group, eds

August 1, 2006

The AIPS++ consortium may be reached by email at aips2-request@nrao.edu. The postal address is: AIPS++ Consortium, c/o NRAO, 520 Edgemont Rd., Charlottesville, Va. 22903-2475 USA.

Contents

Chapter 1

BIMA

BIMA specific modules and tools

include "bima.g"

Description
Package for BIMA-related functionality

1.1 mirfiller - Module

Module for filling MIRIAD data
include "mirfiller.g"

Description

This module contains the `mirfiller` tool, which is used to load data in MIRIAD format in AIPS++ Measurement Sets. See the mirfiller tool documentation for details.

1.1.1 mirfiller - Tool

Package	**bima**
Module	**mirfiller**

a tool for filling MIRIAD data
include "mirfiller.g"

Description

`mirfiller` tool, which is used to load data in MIRIAD format in AIPS++ Measurement Sets. When a `mirfiller` tool is created, it is attached to a Miriad dataset; it can subsequently be used to fill several output measurement sets, perhaps with different data from the input MIRIAD dataset. The actually filling is commenced by calling the fill function. Restricting the filling to specific spectral windows can be controlled via the select function. `mirfiller` tries to be smart about figuring out which spectral windows pass to the output measurment sets by default, depending on what data it finds in the dataset. (At the moment, it tries to decide which wideband channels to pass; see section on MIRIAD layout below.) The default behavior can be changed via the `defpass` parameter given either to the `mirfiller` constructor or the `select()` function. See the description of select for more details.

Example

```
include 'mirfiller.g'

# open a miriad dataset; watch the logger for a summary and
#   mirfiller's guess as to what windows you'll probably want.
frommir := mirfiller('3c273')

# load the windows that mirfiller thinks you'll want.
frommir.fill('3c273.ms')

# load all the windows
frommir.select(defpass='all');
frommir.fill('3c273all.ms');

# load only the sideband averages to an MS called 3c273wide.ms
frommir.select(defpass='none', sbandav='all')
frommir.fill('3c273wide.ms')

# load only the first four spectral line windows and their
#   corresponding averages
frommir.select(defpass='none', splwin=[1:4], winav=[1:4])
frommir.fill('3c273_1-4.ms');
```

MIRIAD Dataset Spectral Layout and its Mapping to the Measurement Set

Each record in a Miriad dataset organizes its correlation data by spectral windows and so-called wideband channels. Each Miriad spectral window contains data from a band of frequencies, broken up into some number of channels. (Because each channel, then, is fairly narrow in bandwidth, these correlations are often referred to as "narrow-band" data.) The wideband channels, in general, contain single correlations, averaged over some band at potentially arbitrary frequencies; that is, they are bolometer-like measurements.

In Miriad datasets written by the BIMA telescope, the wideband channels are used to store wide-band averages of the spectral windows. More specifically, the first wideband channel contains the average of all spectral windows channels associated with the lower sideband (the first *Nwin/2* windows), and the second wideband channel contains the average of the upper sideband windows. The remaining wideband channels contain the averages of each of the spectral windows (in the same order). Thus, if the BIMA Miriad dataset contains *Nwin* spectral windows, there will be *Nwin+2* wideband channels. The first *Nwin/2* windows will be from the lower sideband, and the rest, from the upper sideband.

When the Miriad data is converted to an MS, each spectral window and each wideband channel is loaded into a separate MS spectral window. Thus, for a BIMA dataset with Nwin windows, the output MS (by default) will contain *Nwin*2+2* spectral windows. If all narrow and wideband data are present, `mirfiller` will store the relationships into the output measurement set so that they can be used appropriately in the calibration process. The select function can be used to further control the windows that get written.

Which windows that should get passed by default (i.e. where they are not explicitly specified by the select function) can depend on what "flavor" visibility dataset it is; that is, more specifically, where its been in its processing history and what the user wants to do with it. For example, if the data is calibrated, then the wideband data is usually no longer needed. (Calibrated or uncalibrated, the window averages are usually not needed.) Furthermore, if it has gone through much processing in Miriad, it is possible that some of the windows or wideband channels may have been discarded; thus, the convention for wideband channels described above is no longer valid. Finally, it's possible that the dataset did not originate from the BIMA telescope, and thus, the convention for wideband channels does not apply. `mirfiller` attempts to detect the "flavor" of the input dataset and adjusts its default window selection accordingly. See the select function for details.

Polarization

A Miriad dataset stores correlations for different polarization correlation types (e.g. RR, RL, etc.) in separate records, even if they were observed simultaneously. (Thus, it is up to the Miriad I/O software to properly combine records to form new Stokes correlations.) This is in contrast with AIPS++ which can store polarizations observed at the same time in the same record. Furthermore, BIMA can only observe one polarization per baseline at a time. `mirfiller` by default will load visibilities of different polarizations into separate polarization setups; thus, each out MS record will contain a single polarization. This is most appropriate for uncalibrated BIMA data.

(Normally, the polarization data are edited and calibrated separately; then, the data is smoothed in time to approximate simultaneous observations.) However, when the data contains multiple, simultaneously-observed polarizations–e.g., VLA data or calibrated and time-averaged BIMA data–it is possible to collect the polarizations into a single polarization setup record using the `joinpol` option (see setoptions()). Thus, when `joinpol` is set to True, each MS record will contain the visibilities for all polarizations. One caveat of this option is that data for any polarization not found for a given baseline will be set to zero. Thus, if you use set `joinpol=T` when loading the raw (unaveraged) BIMA data, it will be N times larger than if it were set to false, where N is the number of different polarizations.

mirfiller - Constructor

Package	**bima**
Module	**mirfiller**
Tool	**mirfiller**

Construct a mirfiller tool

Synopsis
mirfiller(mirfile, preview, defpass)

Description
This creates a mirfiller tool attached to a Miriad dataset. The Miriad dataset is opened and, by default, is scanned to determine its contents. A summary of the contents is then sent to the logger, listing useful information such as the number of array configurations found, the source names, and, most importantly, the frequency setups found.
Based on what it finds in the dataset, `mirfiller` will determine what "flavor" Miriad visibility dataset it is. Currently recognized are:

- Uncalibrated BIMA data: this is data that has a frequency layout in convention used by the BIMA telescope. (See "MIRIAD Dataset Spectral Layout and its Mapping to the Measurement Set" section of the mirfiller tool description.)
- Calibrated BIMA data: Currently, this "flavor" is matched for any BIMA dataset that appears to be missing spectral line windows or window averages. The rationale is that this is an indication that the data has been calibrated and the undesired data have been dropped.
- Generic Miriad data: This is any dataset that appears not to follow frequency layout at all.

After giving the summary, `mirfiller` will report which catagory above it believes this dataset belongs. Based on this determination, it will set the default window passing mode as appropriate. This mode can be overridden with via `defpass` parameter, either here or in the `select()` function. The major aim of this mode is to minimize the amount that the user must specify in terms window selection for the most common uses of the data.

Arguments

mirfile	the file name of the Miriad dataset
preview	if true, the file will be scanned in full to determine its contents. It may be helpful to set this to false for very large files. In this case, only the beginning of the file is scanned, and the default defpass value is based on the limited information read. Default: T
defpass	a string indicating the default window passing mode to use. This mode affects the which kinds of windows get passed when they are not explicitly specified by the select() function. Allowed: 'default' – let mirfiller determine which mode to use based on what is found in the data; 'rawbima' – use a default appropriate for uncalibrated BIMA data: pass all spectral line windows and both side band averages but no window averages; 'calbima' – use a default appropriate for calibrated data: pass all spectral line windows but no wideband averages; 'all' – pass all windows and wideband channels by default; 'none' – pass no windows or wideband channels by default, require that they be specified explicitly with select(). Default: 'default'

Returns
mirfiller tool

Example
This example attaches a mirfiller tool to the miriad dataset called 3c273 and assigns it to a Glish variable named `frommir`.

```
frommir := mirfiller('3c273')
```

This causes a summary to be sent to the logger which will look something like this:

```
Miriad UV dataset: 3c273
 Max. no. of visibility records:    990
 Max. no. of spectral line windows: 12 (Max no. of channels: 1280)
 Max. no. of wide-band channels:    14
 No. of array configurations:       1
 No. of polarizations:              1
 Time Range: 1997/11/02/19:44:55 - 1997/11/02/20:03:35
Sources:
     3C273      12:29:06.70     +02.03.08.60 (1 field)
Frequency Setup:
   Mode 6, 1280 line channels, 14 wide channels
 Window  #chans   start freq.     increment    bandwidth    rest freq.
    1      256   93.17534 GHz     -48.83 kHz   12.50 MHz    93.17378 GHz
    2       32   93.11704 GHz   -3125.00 kHz  100.00 MHz    96.41298 GHz
    3       32   93.01704 GHz   -3125.00 kHz  100.00 MHz    96.41298 GHz
    4      256   92.87483 GHz     -48.83 kHz   12.50 MHz    96.41298 GHz
    5       32   92.81705 GHz   -3125.00 kHz  100.00 MHz    96.41298 GHz
    6       32   92.71705 GHz   -3125.00 kHz  100.00 MHz    96.41298 GHz
    7      256   96.10183 GHz      48.83 kHz   12.50 MHz    96.41298 GHz
    8       32   96.16013 GHz    3125.00 kHz  100.00 MHz    96.41298 GHz
    9       32   96.26013 GHz    3125.00 kHz  100.00 MHz    96.41298 GHz
   10      256   96.40233 GHz      48.83 kHz   12.50 MHz    96.41298 GHz
   11       32   96.46011 GHz    3125.00 kHz  100.00 MHz    96.41298 GHz
   12       32   96.56011 GHz    3125.00 kHz  100.00 MHz    96.41298 GHz
Input looks like a raw BIMA dataset; setting default defpass="rawbima"
```

---

mirfiller.select - Function

Package	**bima**
Module	**mirfiller**
Tool	**mirfiller**

select data from the input Miriad dataset.

Synopsis

select(defpass, splwin, winav, sbandav)

Description

Correllation data stored in a Miriad dataset are stored in spectral windows and so-called wideband channels. In BIMA data, the latter are used to store wideband averages. BIMA data contains an average for each window and one for each entire sideband. This function allows one to select which spectral line windows (via `splwin`) and which wideband channels (via `winav` and `sbandav`) to pass.

As a way of minimizing the amount the user has to type in selecting windows, `mirfiller` supports a notion of a *default window passing mode*, which can be adapted to the data found in the Miriad dataset. Controlled via the `defpass` parameter, it specifies for each type of input window or channel what to pass (usually all or none) if it is not specified via the `splwin, winav,` or `sbandav` parameters. See `defpass` below for currently available modes.

Each call to this function completely overrides previously selections.

Arguments

defpass	a string indicating the default window passing mode to use. This mode affects the which kinds of windows get passed when they are not explicitly specified by the other parameters. See general description above for more details. Allowed: 'default' – use the mode that was set (or assumed) at construction (see constructor description for more information); 'rawbima' – use a default appropriate for uncalibrated BIMA data: pass all spectral line windows and both side band averages but no window averages; 'calbima' – use a default appropriate for calibrated data: pass all spectral line windows but no wideband averages; 'all' – pass all windows and wideband channels by default; 'none' – pass no windows or wideband channels by default, require that they be specified explicitly with select(). Default: 'default'
splwin	a list of one-based spectral window indicies to load. Default: [], the defpass parameter controls how this is interpreted
winav	a list of one-based window average indicies to load. An index of 1 means load the average of the first spectral window. Default: [], the defpass parameter controls how this is interpreted
sbandav	a string indicating which sideband averages to load. Allowed: 'default' – allow the defpass parameter to control sideband average selection; 'lsb' – the lower sideband; 'usb' – the upper sideband; 'all' – both sidebands; 'none' – neither sideband. Default: 'default'

Example

```
# open a miriad dataset; watch the logger for a summary and
#   mirfiller's guess as to what windows you'll probably want.
frommir := mirfiller('3c273')

# load the windows that mirfiller thinks you'll want.
frommir.fill('3c273.ms')
```

```
# load all data into '3c273all.ms'
frommir.select(defpass='all');
frommir.fill('3c273all.ms')

# load only the sideband averages to an MS called 3c273wide.ms
frommir.select(defpass='none', sbandav='all')
frommir.fill('3c273wide.ms')

# load only the first four spectral line windows and their
#   corresponding averages
frommir.select(defpass='none', splwin=[1:4], winav=[1:4])
frommir.fill('3c273_1-4.ms');
```

---

mirfiller.summary - Function

Package	**bima**
Module	**mirfiller**
Tool	**mirfiller**

return and/or print a summary of the contents of the Miriad dataset

Synopsis
summary(header, verbose, preview)

Description
This function provides a summary of the contents of the Miriad dataset. By default, this information is printed as logger messages (unless the verbose parameter is false). Normally, this will cause the dataset to be scanned in its entirety, if it hasn't been already, to extract the information; by setting preview=F, the scanning will be prevented, and thus the summary may be based only on the first timestep in the dataset. Note that if the `mirfiller` tool was constructed with `preview=T` (the default) the dataset will be scanned then; thus, the preview parameter subsequently passed to `summary()` function will be ignored (since all the needed information has been gathered).
Some of the information can also be returned as a record via the header parameter. Fields that will be loaded into this record include:

- `nwide` — the maximum number of wide-band channels
- `nspect` — the maximum number of spectral line windows
- `nchan` — the maximum (total) number of channels

- `narray` — the number of array configurations found
- `npol` — the number of polarizations found
- `pols` — an array of the polarizations found
- `starttime` — the UTC starting time in seconds
- `endtime` — the UTC ending time in seconds
- `nrec` — the total number of visibility records found
- `cormode` — the correlator mode of the first correlator setup
- `telescope` — the names of all the telescopes encountered in the dataset.

Arguments

header	a record to hold the summary information Default: [=] (allowing the caller to ignore the returned information)
verbose	If true, detailed information regarding contents will be sent to the logger. Default: T
preview	if true, the entire file will be read (if it wasn't already scanned when the tool was constructed) to extract information. Set this to false to defeat this behavior (e.g. for very large files). In this case, only the first record of the file will be read; thus, some information may be inaccurate if the dataset is a concatonation of several original datasets. Default: T

Example

```
frommir := mirfiller('3c273', preview=F, quiet=T);  # preview=F prevents full scanning
hdr := [=];
frommir.summary(hdr, preview=T)   # preview=T causes scanning to happen now
```

This causes a summary to be sent to the logger which will look something like this:

```
Miriad UV dataset: 3c273
 Max. no. of visibility records:    990
 Max. no. of spectral line windows: 12 (Max no. of channels: 1280)
 Max. no. of wide-band channels:    14
```

```
 No. of array configurations:       1
 No. of polarizations:              1
 Time Range: 1997/11/02/19:44:55 - 1997/11/02/20:03:35
Sources:
     3C273      12:29:06.70     +02.03.08.60 (1 field)
Frequency Setup:
   Mode 6, 1280 line channels, 14 wide channels
 Window  #chans   start freq.     increment    bandwidth    rest freq.
    1      256   93.17534 GHz     -48.83 kHz   12.50 MHz    93.17378 GHz
    2       32   93.11704 GHz   -3125.00 kHz  100.00 MHz    96.41298 GHz
    3       32   93.01704 GHz   -3125.00 kHz  100.00 MHz    96.41298 GHz
    4      256   92.87483 GHz     -48.83 kHz   12.50 MHz    96.41298 GHz
    5       32   92.81705 GHz   -3125.00 kHz  100.00 MHz    96.41298 GHz
    6       32   92.71705 GHz   -3125.00 kHz  100.00 MHz    96.41298 GHz
    7      256   96.10183 GHz      48.83 kHz   12.50 MHz    96.41298 GHz
    8       32   96.16013 GHz    3125.00 kHz  100.00 MHz    96.41298 GHz
    9       32   96.26013 GHz    3125.00 kHz  100.00 MHz    96.41298 GHz
   10      256   96.40233 GHz      48.83 kHz   12.50 MHz    96.41298 GHz
   11       32   96.46011 GHz    3125.00 kHz  100.00 MHz    96.41298 GHz
   12       32   96.56011 GHz    3125.00 kHz  100.00 MHz    96.41298 GHz
Input looks like a raw BIMA dataset; setting default defpass="rawbima"
```

Furthermore, the `hdr` record now holds some information:

```
- print hdr
[nwide=14, nspect=12, nchan=1280, cormode=6, npol=1, pols=YY, starttime=4.38522e+09,
telescope=BIMA, scanned=T, endtime=4.38522e+09, narray=1, nrec=990]
```

---

mirfiller.getoptions - Function

Package	**bima**
Module	**mirfiller**
Tool	**mirfiller**

return the current filler options

Synopsis
getoptions()

Description
`mirfiller` supports a variety of options that can be set to control the filling, and this function returns their state as a Glish record. The currently supported options are:

- `scanlim` — the scan time jump limit (quantity). If the jump in time between two consecutive Miriad records is greater than this limit the scan number that gets written out for that record will be incremented. A change in source will always increment the scan number unless scanlim is negative; in this case, all records are forced to have the same scan number. The default is 5 minutes (`'5min'`).
- `obslim` — the observation ID time jump limit (quantity). The observation ID is meant to delimit two tracks that might appear in the same file. If the jump in time between two consecutive Miriad records is greater than this limit the scan number that gets written out for that record will be incremented. The ID will always be incremented if there is a change in telescope or array configuration unless obslim is negative, in which case, all records will be forced to have the same observation ID. The default is four hours (`'4h'`).
- `tilesize` — the tiling size to use for storing data in the MS using the TiledStorageManager. If the value is ¡= 0, the standard (non-tiled) storage manager will be used. The default is 32.
- `verbose` — If true, extra messages will be sent to the logger.
- `wideconv` — a string representing the convention to use for interpreting the wideband channels. The allowed values are "bima" and "none." In the "bima" convention, the first two wideband channels are the lower & upper sideband averages and the remaining channels represent averages of each of the spectral line windows. Currently, this option only affects the writing out of window associations: if wideconv="bima", the associations linking wideband windows to spectral line windows will be written according to the convention.
- `joinpol` — If false (the default), visibilities of different polarization correlation types will be loaded into different records. This is appropriate for uncalibrated BIMA data. If this option is true, the different polarizations observed at the same time will be gathered into a output record. This is appropriate for BIMA polarization measurement data that has been calibrated and time-averaged or data from a telescope with dual-polarization receivers. See section on Polarization in mirfiller module description for further discussion.
- `tsyswt` — If true (the default), the system temperatures will be used to estimate the theoretical noise for the data which will subsequently be used (automatically) as weights during imaging. If false, system temperature weighting will be disabled. (System temperatures will be loaded regardless.)
- `compress` — If true, the data will be written in compressed form. Compression reduces the size of the output MS by about a factor of two

but also limits the dynamic range to 65000:1 (well above what is achievable with typical BIMA data). The default is currently false.

- `updmodelint` — A time quantity representing how often a new planet model should be written. Default is 8 hours (`'8h'`). For observations of planets, a record is entered into the SOURCE_MODEL column of the SOURCE subtable which points to the planet model. A new model is written after every interval, 'updmodelint' has passed.

Example
The example shows a Glish session demonstrating `getoptions()` and `setoptions()`. Text to the right of the "- " represents the user input.

```
- frommir := mirfiller('3c273')
- frommir.getoptions()
[tilesize=32, scanlim=[value=5, unit=min], obslim=[value=4, unit=h],
wideconv=bima, joinpol=F, tsyswt=T, planetfit=4, movfield=T,
updmodelint=[value=8, unit=h], verbose=F]
- frommir.setoptions(verbose=T, scanlim='2min')
T
- frommir.getoptions()
[tilesize=32, scanlim=[value=2, unit=min], obslim=[value=4, unit=h],
wideconv=bima, joinpol=F, tsyswt=T, planetfit=4, movfield=T,
updmodelint=[value=8, unit=h], verbose=T]
- frommir.setoptions(obslim='0.5h')
T
- frommir.getoptions();
[tilesize=32, scanlim=[value=2, unit=min], obslim=[value=0.5, unit=h],
wideconv=bima, joinpol=F, tsyswt=T, planetfit=4, movfield=T,
updmodelint=[value=8, unit=h], verbose=T]
- frommir.setoptions(updmodelint='30min');
T
- frommir.getoptions();
[tilesize=32, scanlim=[value=5, unit=min], obslim=[value=4, unit=h],
wideconv=bima, joinpol=F, tsyswt=T, planetfit=4, movfield=T,
updmodelint=[value=0.5, unit=h], verbose=F]
- frommir.setoptions(verbose=T, reset=T)
T
- frommir.getoptions();
[tilesize=32, scanlim=[value=5, unit=min], obslim=[value=4, unit=h],
wideconv=none, joinpol=F, tsyswt=T, planetfit=4, movfield=T,
updmodelint=[value=8, unit=h], verbose=T]
```

---

mirfiller.setoptions - Function

Package	**bima**
Module	**mirfiller**
Tool	**mirfiller**

return the current filler options

Synopsis
setoptions(scanlim, obslim, tilesize, verbose, wideconv, joinpol, tsyswt, compress, updmodelint, reset)

Description
This function allows one to set a number of options that control the filling. Normally, if an option value is not provided, the option will retain its previously set value (or its default if it has not been set by the user). This allows one to set options independently. This behavior can be overridden with `reset=T` which will reset all option values to their default before the newly provided values are set.

Arguments

scanlim	the scan time jump limit. If the jump in time between two consecutive Miriad records is greater than this limit the scan number that gets written out for that record will be incremented. A change in source will always increment the scan number unless scanlim is negative; in this case, all records are forced to have the same scan number. Default: 5 minutes
obslim	the observation ID time jump limit. The observation ID is meant to delimit two tracks that might appear in the same file. If the jump in time between two consecutive Miriad records is greater than this limit the scan number that gets written out for that record will be incremented. The ID will always be incremented if there is a change in telescope or array configuration unless obslim is negative, in which case, all records will be forced to have the same observation ID. Default: 5 minutes
tilesize	the tiling size to use for storing data in the MS using the TiledStorageManager. If the value is ¡= 0, the standard (non-tiled) storage manager will be used. Default: 32
verbose	If true, extra messages will be sent to the logger. Default: F
wideconv	a string representing the convention to use for interpreting the wideband channels. Currently, this option only affects the writing out of window associations in the spectral window subtable. Allowed: 'bima' – assume that the first two wideband channels are the lower & upper sideband averages and the remaining channels represent averages of each of the spectral line windows; the output wideband and spectral line windows will be linked accordingly in the spectral window subtable. 'none' – assume no convention for the wideband channels, and therefore, encode no associations. Default: the convention detected when mirfiller was constructed
joinpol	If false (the default), visibilities of different polarization correlation types will be loaded into different records. This is appropriate for uncalibrated BIMA data. If this option is true, the different polarizations observed at the same time will be gathered into a output record. This is appropriate for BIMA polarization measurement data that has been calibrated and time-averaged or data from a telescope with dual-polarization receivers. See section on Polarization in mirfiller module description for further discussion. Default: F
tsyswt	If true (the default), the system temperatures will be used to estimate the theoretical noise for the data which will subsequently be used (automatically) as weights during imaging. If false, system temperature weighting will be disabled. (System temperatures will be loaded regardless.) Default: T
compress	If true, the data will be written in compressed form. Compression reduces the size of the output MS by about a factor of two but also limits the dynamic range to 65000:1 (well above what is achievable with typical BIMA data). Default: F
updmodelint	the time interval after which the model for planet observations should be updated. Default: 8 hours
reset	If true, all above options are reset to their default values before modifying them to the new values Default: F

Example

The example shows a Glish session demonstrating `getoptions()` and `setoptions()`. Text to the right of the "- " (and before the "#") represents the user input; the text after "#" explain what's happening.

```
- frommir := mirfiller('3c273')
- frommir.getoptions()                      # see the default options
[tilesize=32, scanlim=[value=5, unit=min], obslim=[value=4, unit=h],
wideconv=none, joinpol=F, tsyswt=T, planetfit=4, movfield=T,
updmodelint=[value=8, unit=h], verbose=T]
- frommir.setoptions(verbose=T, scanlim='2min') # set two options
T
- frommir.getoptions();                     # see the results
[tilesize=32, scanlim=[value=2, unit=min], obslim=[value=4, unit=h],
wideconv=none, joinpol=F, tsyswt=T, planetfit=4, movfield=T,
updmodelint=[value=8, unit=h], verbose=T]
- frommir.setoptions(obslim='0.5h')            # set an additional option
T
- frommir.getoptions()
[tilesize=32, scanlim=[value=120, unit=min], obslim=[value=0.5, unit=h],
wideconv=none, joinpol=F, tsyswt=T, planetfit=4, movfield=T,
updmodelint=[value=8, unit=h], verbose=T]
- frommir.setoptions(updmodelint='120min') # set model update interval to 120 minutes
T
- frommir.getoptions()
[tilesize=32, scanlim=[value=120, unit=min], obslim=[value=0.5, unit=h],
wideconv=none, joinpol=F, tsyswt=T, planetfit=4, movfield=T,
updmodelint=[value=2, unit=h], verbose=T]
- frommir.setoptions(verbose=T, reset=T)    # now reset and set an option
T
- frommir.getoptions();
[tilesize=32, scanlim=[value=5, unit=min], obslim=[value=4, unit=h],
wideconv=none, joinpol=F, tsyswt=T, planetfit=4, movfield=T,
updmodelint=[value=8, unit=h], verbose=T]
```

---

mirfiller.fill - Function

Package	**bima**
Module	**mirfiller**
Tool	**mirfiller**

write selected data into a measurement set

Synopsis
fill(msfile, verbose)

Description
This function does the actual work of converting the Miriad dataset to a Measurment Set. A measurement set with the given name will be created and filled with data (previously selected with the select function) from the input Miriad dataset. When the filling is complete, a summary of the number of visibilities, spectral windows, fields, and polarizations will be written to the logger.
The fill function can be called several times, usually between calls to select. This allows one to write different parts of the Miriad dataset into different output datasets.
Enabling the verbose option (via the setoptions function) will cause `fill()` to print extra messages about the data it finds in the Miriad dataset. In particular, it will print out the timestamp for each new scan. (A scan is a group of integrations taken close together in time. What exactly constitutes a scan is controlled by the scanlim option; see setoptions for details.)

Arguments

msfile	the file name of the ouput MS dataset
verbose	If true, extra messages will be sent to the logger during this fill process. Setting this to true is equivalent to running setoptions(verbose=T) just before calling fill() and then setting it back again after fill() is finished. If the verbose option has been turned on via setoptions(), then extra messages are sent regardless of the parameter given here. Default: F

Example

```
# open a miriad dataset
frommir := mirfiller('3c273')

# load all data into '3c273all.ms'
frommir.fill('3c273all.ms')

# load only the sideband averages to an MS called 3c273wide.ms
frommir.select(splwin='none', winav='none')
frommir.fill('3c273wide.ms')
```

```
frommir.done();
```

mirfiller.done - Function

Package	**bima**
Module	**mirfiller**
Tool	**mirfiller**

shut down this filler

Synopsis
done()

Description
This will close the input Miriad dataset and shut down this tool. If this is the last open mirfiller tool open, the server will be shutdown.

Example

```
- frommir := mirfiller('3c273')
- frommir.done()                  # oops.
F
- print frommir                   # frommir is no longer a mirfiller tool
F
```

1.1.2 mirfillertest - Function

Package	**bima**
Module	**mirfiller**

Function to test mirfiller

Synopsis
mirfillertest(summary, details, logsummary, verbose, fill, mirdata, modeldata)

Description

`mirfillertest` is a function to test mirfiller. By default, it uses datasets in the AIPS++ data repository, although the user can can specify other datasets to use for the test. The basic strategy of the tests is to fill a `Miriad` dataset into MS2 format and then to test the MS against a fiducial MS that has been filled by mirfiller in the past. Tests include comparing column names, column keywords, and column data for the main MS table as well as all the associated subtables (except the HISTORY subtable). The function returns `T` if all tests pass, `F` otherwise. The tests (including filling) take 15 seconds on an unloaded Linux system with a 1.5 GHz Pentium 4 processor. The function will create a scratch directory in the current directory called `mirfillertester.work` where it will fill the `Miriad` dataset. For the default dataset, this will require about 5.5 MB of disk space.

Arguments

summary	out	Record in which to store test results Allowed:Any valid record Default: [=]
details	out	Record in which to store column data when column tests fail Allowed:Any valid record Default: [=]
logsummary	in	Send a summary of the results to the logger? Allowed:Boolean Default: F
verbose	in	Controls the amount of logger output, 0=none, higher numbers=more Allowed:Int ¿= 0 Default: 1
fill	in	Fill data? Allowed:Boolean Default: T
mirdata	in	Miriad dataset to fill Allowed:Any valid Miriad dataset name Default: " (defaults to dataset in data repository)
modeldata	in	Fiducial measurement set for comparison Allowed:Any valid measurement set name Default: " (defaults to dataset in data repository)

Returns
Boolean or fail

Example

```
- include 'mirfiller.g'
- mft_summary := [=];
- mft_details := [=];
# run the tests, being somewhat verbose putting results in mft_summary and
# mft_details, and have a summary sent to the logger
- mirfillertest(mft_summary,mft_details,T);
NORMAL: defaultos (dos) ready
NORMAL: table system ready
NORMAL: Begin filling  to mirfillertester.work/sgrb2n.test.ms
NORMAL: Starting server mirfiller
NORMAL: Server started: /appl/aips++/weekly/linux_gnu/bin/mirfiller
(AIPS++ version: 1.7 (build #173))
NORMAL:
Summary of Miriad UV dataset:
/appl/aips++/weekly/data/bima/dev/miriad/sgrb2n.small.ll
 Max. no. of visibility records:    396
 Max. no. of spectral line windows: 8 (Max no. of channels: 1024)
 Max. no. of wide-band channels:    10
 No. of array configurations:       1
 Telescopes: BIMA
 Polarizations found: YY
 Time Range: 1997/11/08/20:18:38 - 1997/11/08/20:22:41
Sources:
    SGRB2N      17:47:19.79     -28.22.17.30 (1 field)
Frequency Setup:
   Mode 4, 1024 line channels, 10 wide channels
 Window  #chans   start freq.     increment    bandwidth    rest freq.
    1      128   109.84321 GHz    -390.62 kHz   50.00 MHz    109.84363
    GHz
    2      128   109.67824 GHz    -390.62 kHz   50.00 MHz    109.68034
    GHz
    3      128   109.59827 GHz    -390.62 kHz   50.00 MHz    109.59875
    GHz
    4      128   109.29134 GHz    -390.62 kHz   50.00 MHz    109.40000
    GHz
    5      128   112.78287 GHz     390.62 kHz   50.00 MHz    112.83351
    GHz
    6      128   112.94784 GHz     390.62 kHz   50.00 MHz    113.00000
    GHz
    7      128   113.02781 GHz     390.62 kHz   50.00 MHz    113.00000
    GHz
    8      128   113.33475 GHz     390.62 kHz   50.00 MHz    113.38305
    GHz
NORMAL: Input looks like a raw BIMA dataset; setting default
```

```
defpass="rawbima"
NORMAL: Starting mirfiller::fill
NORMAL: Accepted 396 input MIRIAD records.
Loaded 7128 data records for
       409464 visibilities,
       18 spectral windows,
       1 polarization,
       1 field, and
       1 array configuration.
NORMAL: Finished mirfiller::fill
       2.57 real        1.31 user        1.21 system
NORMAL: successful readonly open of default-locked table
mirfillertester.work/sgrb2n.test.ms: 22 columns, 7128 rows
NORMAL: successful readonly open of default-locked table
/appl/aips++/data/bima/dev/ms/sgrb2n.ll.small.ms: 22 columns, 7128 rows
NORMAL: successful readonly open of default-locked table
mirfillertester.work/sgrb2n.test.ms/ANTENNA: 8 columns, 10 rows
NORMAL: successful readonly open of default-locked table
/appl/aips++/data/bima/dev/ms/sgrb2n.ll.small.ms/ANTENNA: 8 columns, 10 rows
ANTENNA
NORMAL: successful readonly open of default-locked table
mirfillertester.work/sgrb2n.test.ms/DATA_DESCRIPTION: 3 columns, 18
rows
NORMAL: successful readonly open of default-locked table
/appl/aips++/data/bima/dev/ms/sgrb2n.ll.small.ms/DATA_DESCRIPTION: 3 columns, 18 rows
DATA_DESCRIPTION
NORMAL: successful readonly open of default-locked table
mirfillertester.work/sgrb2n.test.ms/FEED: 12 columns, 10 rows
NORMAL: successful readonly open of default-locked table
/appl/aips++/data/bima/dev/ms/sgrb2n.ll.small.ms/FEED: 12 columns, 10 rows
FEED
NORMAL: successful readonly open of default-locked table
mirfillertester.work/sgrb2n.test.ms/FLAG_CMD: 8 columns, 0 rows
NORMAL: successful readonly open of default-locked table
/appl/aips++/data/bima/dev/ms/sgrb2n.ll.small.ms/FLAG_CMD: 8 columns, 0 rows
FLAG_CMD
NORMAL: successful readonly open of default-locked table
mirfillertester.work/sgrb2n.test.ms/FIELD: 9 columns, 1 rows
NORMAL: successful readonly open of default-locked table
/appl/aips++/data/bima/dev/ms/sgrb2n.ll.small.ms/FIELD: 9 columns, 1 rows
FIELD
NORMAL: successful readonly open of default-locked table
mirfillertester.work/sgrb2n.test.ms/OBSERVATION: 9 columns, 1 rows
NORMAL: successful readonly open of default-locked table
/appl/aips++/data/bima/dev/ms/sgrb2n.ll.small.ms/OBSERVATION: 9 columns, 1 rows
OBSERVATION
```

```
NORMAL: successful readonly open of default-locked table
mirfillertester.work/sgrb2n.test.ms/POINTING: 9 columns, 0 rows
NORMAL: successful readonly open of default-locked table
/appl/aips++/data/bima/dev/ms/sgrb2n.ll.small.ms/POINTING: 9 columns, 0 rows
POINTING
NORMAL: successful readonly open of default-locked table
mirfillertester.work/sgrb2n.test.ms/POLARIZATION: 4 columns, 1 rows
NORMAL: successful readonly open of default-locked table
/appl/aips++/data/bima/dev/ms/sgrb2n.ll.small.ms/POLARIZATION: 4 columns, 1 rows
POLARIZATION
NORMAL: successful readonly open of default-locked table
mirfillertester.work/sgrb2n.test.ms/PROCESSOR: 5 columns, 1 rows
NORMAL: successful readonly open of default-locked table
/appl/aips++/data/bima/dev/ms/sgrb2n.ll.small.ms/PROCESSOR: 5 columns, 1 rows
PROCESSOR
NORMAL: successful readonly open of default-locked table
mirfillertester.work/sgrb2n.test.ms/SPECTRAL_WINDOW: 17 columns, 18
rows
NORMAL: successful readonly open of default-locked table
/appl/aips++/data/bima/dev/ms/sgrb2n.ll.small.ms/SPECTRAL_WINDOW: 17 columns, 18 rows
SPECTRAL_WINDOW
NORMAL: successful readonly open of default-locked table
mirfillertester.work/sgrb2n.test.ms/STATE: 7 columns, 1 rows
NORMAL: successful readonly open of default-locked table
/appl/aips++/data/bima/dev/ms/sgrb2n.ll.small.ms/STATE: 7 columns, 1 rows
STATE
NORMAL: successful readonly open of default-locked table
mirfillertester.work/sgrb2n.test.ms/SOURCE: 12 columns, 1 rows
NORMAL: successful readonly open of default-locked table
/appl/aips++/data/bima/dev/ms/sgrb2n.ll.small.ms/SOURCE: 12 columns, 1 rows
SOURCE
NORMAL: successful readonly open of default-locked table
mirfillertester.work/sgrb2n.test.ms/DOPPLER: 4 columns, 8 rows
NORMAL: successful readonly open of default-locked table
/appl/aips++/data/bima/dev/ms/sgrb2n.ll.small.ms/DOPPLER: 4 columns, 8 rows
DOPPLER
NORMAL: successful readonly open of default-locked table
mirfillertester.work/sgrb2n.test.ms/SYSCAL: 8 columns, 180 rows
NORMAL: successful readonly open of default-locked table
/appl/aips++/data/bima/dev/ms/sgrb2n.ll.small.ms/SYSCAL: 8 columns, 180 rows
SYSCAL
NORMAL: All tests passed
NORMAL: T
NORMAL: MAIN.COLNAMES: T
NORMAL: MAIN.COLKEYWORDNAMES.UVW: T
NORMAL: MAIN.COLKEYWORDNAMES.FLAG: T
```

```
NORMAL: MAIN.COLKEYWORDNAMES.FLAG_CATEGORY: T
NORMAL: MAIN.COLKEYWORDNAMES.WEIGHT: T
NORMAL: MAIN.COLKEYWORDNAMES.SIGMA: T
NORMAL: MAIN.COLKEYWORDNAMES.ANTENNA1: T
NORMAL: MAIN.COLKEYWORDNAMES.ANTENNA2: T
NORMAL: MAIN.COLKEYWORDNAMES.ARRAY_ID: T
NORMAL: MAIN.COLKEYWORDNAMES.DATA_DESC_ID: T
NORMAL: MAIN.COLKEYWORDNAMES.EXPOSURE: T
NORMAL: MAIN.COLKEYWORDNAMES.FEED1: T
NORMAL: MAIN.COLKEYWORDNAMES.FEED2: T
NORMAL: MAIN.COLKEYWORDNAMES.FIELD_ID: T
NORMAL: MAIN.COLKEYWORDNAMES.FLAG_ROW: T
NORMAL: MAIN.COLKEYWORDNAMES.INTERVAL: T
NORMAL: MAIN.COLKEYWORDNAMES.OBSERVATION_ID: T
NORMAL: MAIN.COLKEYWORDNAMES.PROCESSOR_ID: T
NORMAL: MAIN.COLKEYWORDNAMES.SCAN_NUMBER: T
NORMAL: MAIN.COLKEYWORDNAMES.STATE_ID: T
NORMAL: MAIN.COLKEYWORDNAMES.TIME: T
NORMAL: MAIN.COLKEYWORDNAMES.TIME_CENTROID: T
NORMAL: MAIN.COLKEYWORDNAMES.DATA: T
NORMAL: MAIN.COLVALUES.UVW: T
NORMAL: MAIN.COLVALUES.FLAG: T
NORMAL: MAIN.COLVALUES.FLAG_CATEGORY: T
NORMAL: MAIN.COLVALUES.WEIGHT: T
NORMAL: MAIN.COLVALUES.SIGMA: T
NORMAL: MAIN.COLVALUES.ANTENNA1: T
NORMAL: MAIN.COLVALUES.ANTENNA2: T
NORMAL: MAIN.COLVALUES.ARRAY_ID: T
NORMAL: MAIN.COLVALUES.DATA_DESC_ID: T
NORMAL: MAIN.COLKEYWORDNAMES.EXPOSURE: T
NORMAL: MAIN.COLKEYWORDNAMES.FEED1: T
NORMAL: MAIN.COLKEYWORDNAMES.FEED2: T
NORMAL: MAIN.COLKEYWORDNAMES.FIELD_ID: T
NORMAL: MAIN.COLKEYWORDNAMES.FLAG_ROW: T
NORMAL: MAIN.COLKEYWORDNAMES.INTERVAL: T
NORMAL: MAIN.COLKEYWORDNAMES.OBSERVATION_ID: T
NORMAL: MAIN.COLKEYWORDNAMES.PROCESSOR_ID: T
NORMAL: MAIN.COLKEYWORDNAMES.SCAN_NUMBER: T
NORMAL: MAIN.COLKEYWORDNAMES.STATE_ID: T
NORMAL: MAIN.COLKEYWORDNAMES.TIME: T
NORMAL: MAIN.COLKEYWORDNAMES.TIME_CENTROID: T
NORMAL: MAIN.COLKEYWORDNAMES.DATA: T
NORMAL: MAIN.COLVALUES.UVW: T
NORMAL: MAIN.COLVALUES.FLAG: T
NORMAL: MAIN.COLVALUES.FLAG_CATEGORY: T
NORMAL: MAIN.COLVALUES.WEIGHT: T
```

```
NORMAL: MAIN.COLVALUES.SIGMA: T
NORMAL: MAIN.COLVALUES.ANTENNA1: T
NORMAL: MAIN.COLVALUES.ANTENNA2: T
NORMAL: MAIN.COLVALUES.ARRAY_ID: T
NORMAL: MAIN.COLVALUES.DATA_DESC_ID: T
NORMAL: MAIN.COLVALUES.EXPOSURE: T
NORMAL: MAIN.COLVALUES.FEED1: T
NORMAL: MAIN.COLVALUES.FEED2: T
NORMAL: MAIN.COLVALUES.FIELD_ID: T
NORMAL: MAIN.COLVALUES.FLAG_ROW: T
NORMAL: MAIN.COLVALUES.INTERVAL: T
NORMAL: MAIN.COLVALUES.OBSERVATION_ID: T
NORMAL: MAIN.COLVALUES.PROCESSOR_ID: T
NORMAL: MAIN.COLVALUES.SCAN_NUMBER: T
NORMAL: MAIN.COLVALUES.STATE_ID: T
NORMAL: MAIN.COLVALUES.TIME: T
NORMAL: MAIN.COLVALUES.TIME_CENTROID: T
NORMAL: MAIN.COLVALUES.DATA: T
NORMAL: MAIN.KEYWORDFIELDNAMES: T
NORMAL: ANTENNA.COLNAMES: T
NORMAL: ANTENNA.COLKEYWORDNAMES.OFFSET: T
NORMAL: ANTENNA.COLKEYWORDNAMES.POSITION: T
NORMAL: ANTENNA.COLKEYWORDNAMES.TYPE: T
NORMAL: ANTENNA.COLKEYWORDNAMES.DISH_DIAMETER: T
NORMAL: ANTENNA.COLKEYWORDNAMES.FLAG_ROW: T
NORMAL: ANTENNA.COLKEYWORDNAMES.MOUNT: T
NORMAL: ANTENNA.COLKEYWORDNAMES.NAME: T
NORMAL: ANTENNA.COLKEYWORDNAMES.STATION: T
NORMAL: ANTENNA.COLVALUES.OFFSET: T
NORMAL: ANTENNA.COLVALUES.POSITION: T
NORMAL: ANTENNA.COLVALUES.TYPE: T
NORMAL: ANTENNA.COLVALUES.DISH_DIAMETER: T
NORMAL: ANTENNA.COLVALUES.FLAG_ROW: T
NORMAL: ANTENNA.COLVALUES.MOUNT: T
NORMAL: ANTENNA.COLVALUES.NAME: T
NORMAL: ANTENNA.COLVALUES.STATION: T
NORMAL: ANTENNA.KEYWORDFIELDNAMES: T
NORMAL: DATA_DESCRIPTION.COLNAMES: T
NORMAL: DATA_DESCRIPTION.COLKEYWORDNAMES.FLAG_ROW: T
NORMAL: DATA_DESCRIPTION.COLKEYWORDNAMES.POLARIZATION_ID: T
NORMAL: DATA_DESCRIPTION.COLKEYWORDNAMES.SPECTRAL_WINDOW_ID: T
NORMAL: DATA_DESCRIPTION.COLVALUES.FLAG_ROW: T
NORMAL: DATA_DESCRIPTION.COLVALUES.POLARIZATION_ID: T
NORMAL: DATA_DESCRIPTION.COLVALUES.SPECTRAL_WINDOW_ID: T
NORMAL: DATA_DESCRIPTION.KEYWORDFIELDNAMES: T
NORMAL: FEED.COLNAMES: T
```

```
NORMAL: FEED.COLKEYWORDNAMES.POSITION: T
NORMAL: FEED.COLKEYWORDNAMES.BEAM_OFFSET: T
NORMAL: FEED.COLKEYWORDNAMES.POLARIZATION_TYPE: T
NORMAL: FEED.COLKEYWORDNAMES.POL_RESPONSE: T
NORMAL: FEED.COLKEYWORDNAMES.RECEPTOR_ANGLE: T
NORMAL: FEED.COLKEYWORDNAMES.ANTENNA_ID: T
NORMAL: FEED.COLKEYWORDNAMES.BEAM_ID: T
NORMAL: FEED.COLKEYWORDNAMES.FEED_ID: T
NORMAL: FEED.COLKEYWORDNAMES.INTERVAL: T
NORMAL: FEED.COLKEYWORDNAMES.NUM_RECEPTORS: T
NORMAL: FEED.COLKEYWORDNAMES.SPECTRAL_WINDOW_ID: T
NORMAL: FEED.COLKEYWORDNAMES.TIME: T
NORMAL: FEED.COLVALUES.POSITION: T
NORMAL: FEED.COLVALUES.BEAM_OFFSET: T
NORMAL: FEED.COLVALUES.POLARIZATION_TYPE: T
NORMAL: FEED.COLVALUES.POL_RESPONSE: T
NORMAL: FEED.COLVALUES.RECEPTOR_ANGLE: T
NORMAL: FEED.COLVALUES.ANTENNA_ID: T
NORMAL: FEED.COLVALUES.BEAM_ID: T
NORMAL: FEED.COLVALUES.FEED_ID: T
NORMAL: FEED.COLVALUES.INTERVAL: T
NORMAL: FEED.COLVALUES.NUM_RECEPTORS: T
NORMAL: FEED.COLVALUES.SPECTRAL_WINDOW_ID: T
NORMAL: FEED.COLVALUES.TIME: T
NORMAL: FEED.KEYWORDFIELDNAMES: T
NORMAL: FLAG_CMD.COLNAMES: T
NORMAL: FLAG_CMD.COLKEYWORDNAMES.APPLIED: T
NORMAL: FLAG_CMD.COLKEYWORDNAMES.COMMAND: T
NORMAL: FLAG_CMD.COLKEYWORDNAMES.INTERVAL: T
NORMAL: FLAG_CMD.COLKEYWORDNAMES.LEVEL: T
NORMAL: FLAG_CMD.COLKEYWORDNAMES.REASON: T
NORMAL: FLAG_CMD.COLKEYWORDNAMES.SEVERITY: T
NORMAL: FLAG_CMD.COLKEYWORDNAMES.TIME: T
NORMAL: FLAG_CMD.COLKEYWORDNAMES.TYPE: T
NORMAL: FLAG_CMD.COLVALUES.APPLIED: T
NORMAL: FLAG_CMD.COLVALUES.COMMAND: T
NORMAL: FLAG_CMD.COLVALUES.INTERVAL: T
NORMAL: FLAG_CMD.COLVALUES.LEVEL: T
NORMAL: FLAG_CMD.COLVALUES.REASON: T
NORMAL: FLAG_CMD.COLVALUES.SEVERITY: T
NORMAL: FLAG_CMD.COLVALUES.TIME: T
NORMAL: FLAG_CMD.COLVALUES.TYPE: T
NORMAL: FLAG_CMD.KEYWORDFIELDNAMES: T
NORMAL: FIELD.COLNAMES: T
NORMAL: FIELD.COLKEYWORDNAMES.DELAY_DIR: T
NORMAL: FIELD.COLKEYWORDNAMES.PHASE_DIR: T
```

```
NORMAL: FIELD.COLKEYWORDNAMES.REFERENCE_DIR: T
NORMAL: FIELD.COLKEYWORDNAMES.CODE: T
NORMAL: FIELD.COLKEYWORDNAMES.FLAG_ROW: T
NORMAL: FIELD.COLKEYWORDNAMES.NAME: T
NORMAL: FIELD.COLKEYWORDNAMES.NUM_POLY: T
NORMAL: FIELD.COLKEYWORDNAMES.SOURCE_ID: T
NORMAL: FIELD.COLKEYWORDNAMES.TIME: T
NORMAL: FIELD.COLVALUES.DELAY_DIR: T
NORMAL: FIELD.COLVALUES.PHASE_DIR: T
NORMAL: FIELD.COLVALUES.REFERENCE_DIR: T
NORMAL: FIELD.COLVALUES.CODE: T
NORMAL: FIELD.COLVALUES.FLAG_ROW: T
NORMAL: FIELD.COLVALUES.NAME: T
NORMAL: FIELD.COLVALUES.NUM_POLY: T
NORMAL: FIELD.COLVALUES.SOURCE_ID: T
NORMAL: FIELD.COLVALUES.TIME: T
NORMAL: FIELD.KEYWORDFIELDNAMES: T
NORMAL: OBSERVATION.COLNAMES: T
NORMAL: OBSERVATION.COLKEYWORDNAMES.TIME_RANGE: T
NORMAL: OBSERVATION.COLKEYWORDNAMES.LOG: T
NORMAL: OBSERVATION.COLKEYWORDNAMES.SCHEDULE: T
NORMAL: OBSERVATION.COLKEYWORDNAMES.FLAG_ROW: T
NORMAL: OBSERVATION.COLKEYWORDNAMES.OBSERVER: T
NORMAL: OBSERVATION.COLKEYWORDNAMES.PROJECT: T
NORMAL: OBSERVATION.COLKEYWORDNAMES.RELEASE_DATE: T
NORMAL: OBSERVATION.COLKEYWORDNAMES.SCHEDULE_TYPE: T
NORMAL: OBSERVATION.COLKEYWORDNAMES.TELESCOPE_NAME: T
NORMAL: OBSERVATION.COLVALUES.TIME_RANGE: T
NORMAL: OBSERVATION.COLVALUES.LOG: T
NORMAL: OBSERVATION.COLVALUES.FLAG_ROW: T
NORMAL: OBSERVATION.COLVALUES.OBSERVER: T
NORMAL: OBSERVATION.COLVALUES.PROJECT: T
NORMAL: OBSERVATION.COLVALUES.RELEASE_DATE: T
NORMAL: OBSERVATION.COLVALUES.SCHEDULE_TYPE: T
NORMAL: OBSERVATION.COLVALUES.TELESCOPE_NAME: T
NORMAL: OBSERVATION.KEYWORDFIELDNAMES: T
NORMAL: POINTING.COLNAMES: T
NORMAL: POINTING.COLKEYWORDNAMES.DIRECTION: T
NORMAL: POINTING.COLKEYWORDNAMES.ANTENNA_ID: T
NORMAL: POINTING.COLKEYWORDNAMES.INTERVAL: T
NORMAL: POINTING.COLKEYWORDNAMES.NAME: T
NORMAL: POINTING.COLKEYWORDNAMES.NUM_POLY: T
NORMAL: POINTING.COLKEYWORDNAMES.TARGET: T
NORMAL: POINTING.COLKEYWORDNAMES.TIME: T
NORMAL: POINTING.COLKEYWORDNAMES.TIME_ORIGIN: T
NORMAL: POINTING.COLKEYWORDNAMES.TRACKING: T
```

```
NORMAL: POINTING.COLVALUES.DIRECTION: T
NORMAL: POINTING.COLVALUES.ANTENNA_ID: T
NORMAL: POINTING.COLVALUES.INTERVAL: T
NORMAL: POINTING.COLVALUES.NAME: T
NORMAL: POINTING.COLVALUES.NUM_POLY: T
NORMAL: POINTING.COLVALUES.TARGET: T
NORMAL: POINTING.COLVALUES.TIME: T
NORMAL: POINTING.COLVALUES.TIME_ORIGIN: T
NORMAL: POINTING.COLVALUES.TRACKING: T
NORMAL: POINTING.KEYWORDFIELDNAMES: T
NORMAL: POLARIZATION.COLNAMES: T
NORMAL: POLARIZATION.COLKEYWORDNAMES.CORR_TYPE: T
NORMAL: POLARIZATION.COLKEYWORDNAMES.CORR_PRODUCT: T
NORMAL: POLARIZATION.COLKEYWORDNAMES.FLAG_ROW: T
NORMAL: POLARIZATION.COLKEYWORDNAMES.NUM_CORR: T
NORMAL: POLARIZATION.COLVALUES.CORR_TYPE: T
NORMAL: POLARIZATION.COLVALUES.CORR_PRODUCT: T
NORMAL: POLARIZATION.COLVALUES.FLAG_ROW: T
NORMAL: POLARIZATION.COLVALUES.NUM_CORR: T
NORMAL: POLARIZATION.KEYWORDFIELDNAMES: T
NORMAL: PROCESSOR.COLNAMES: T
NORMAL: PROCESSOR.COLKEYWORDNAMES.FLAG_ROW: T
NORMAL: PROCESSOR.COLKEYWORDNAMES.MODE_ID: T
NORMAL: PROCESSOR.COLKEYWORDNAMES.TYPE: T
NORMAL: PROCESSOR.COLKEYWORDNAMES.TYPE_ID: T
NORMAL: PROCESSOR.COLKEYWORDNAMES.SUB_TYPE: T
NORMAL: PROCESSOR.COLVALUES.FLAG_ROW: T
NORMAL: PROCESSOR.COLVALUES.MODE_ID: T
NORMAL: PROCESSOR.COLVALUES.TYPE: T
NORMAL: PROCESSOR.COLVALUES.TYPE_ID: T
NORMAL: PROCESSOR.COLVALUES.SUB_TYPE: T
NORMAL: PROCESSOR.KEYWORDFIELDNAMES: T
NORMAL: SPECTRAL_WINDOW.COLNAMES: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.MEAS_FREQ_REF: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.CHAN_FREQ: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.REF_FREQUENCY: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.CHAN_WIDTH: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.EFFECTIVE_BW: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.RESOLUTION: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.FLAG_ROW: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.FREQ_GROUP: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.FREQ_GROUP_NAME: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.IF_CONV_CHAIN: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.NAME: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.NET_SIDEBAND: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.NUM_CHAN: T
```

```
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.TOTAL_BANDWIDTH: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.ASSOC_SPW_ID: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.ASSOC_NATURE: T
NORMAL: SPECTRAL_WINDOW.COLKEYWORDNAMES.DOPPLER_ID: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.MEAS_FREQ_REF: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.CHAN_FREQ: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.REF_FREQUENCY: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.CHAN_WIDTH: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.EFFECTIVE_BW: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.RESOLUTION: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.FLAG_ROW: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.FREQ_GROUP: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.FREQ_GROUP_NAME: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.IF_CONV_CHAIN: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.NAME: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.NET_SIDEBAND: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.NUM_CHAN: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.TOTAL_BANDWIDTH: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.ASSOC_SPW_ID: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.ASSOC_NATURE: T
NORMAL: SPECTRAL_WINDOW.COLVALUES.DOPPLER_ID: T
NORMAL: SPECTRAL_WINDOW.KEYWORDFIELDNAMES: T
NORMAL: STATE.COLNAMES: T
NORMAL: STATE.COLKEYWORDNAMES.CAL: T
NORMAL: STATE.COLKEYWORDNAMES.FLAG_ROW: T
NORMAL: STATE.COLKEYWORDNAMES.LOAD: T
NORMAL: STATE.COLKEYWORDNAMES.OBS_MODE: T
NORMAL: STATE.COLKEYWORDNAMES.REF: T
NORMAL: STATE.COLKEYWORDNAMES.SIG: T
NORMAL: STATE.COLKEYWORDNAMES.SUB_SCAN: T
NORMAL: STATE.COLVALUES.CAL: T
NORMAL: STATE.COLVALUES.FLAG_ROW: T
NORMAL: STATE.COLVALUES.LOAD: T
NORMAL: STATE.COLVALUES.OBS_MODE: T
NORMAL: STATE.COLVALUES.REF: T
NORMAL: STATE.COLVALUES.SIG: T
NORMAL: STATE.COLVALUES.SUB_SCAN: T
NORMAL: STATE.KEYWORDFIELDNAMES: T
NORMAL: SOURCE.COLNAMES: T
NORMAL: SOURCE.COLKEYWORDNAMES.POSITION: T
NORMAL: SOURCE.COLKEYWORDNAMES.DIRECTION: T
NORMAL: SOURCE.COLKEYWORDNAMES.PROPER_MOTION: T
NORMAL: SOURCE.COLKEYWORDNAMES.CALIBRATION_GROUP: T
NORMAL: SOURCE.COLKEYWORDNAMES.CODE: T
NORMAL: SOURCE.COLKEYWORDNAMES.INTERVAL: T
NORMAL: SOURCE.COLKEYWORDNAMES.NAME: T
```

```
NORMAL: SOURCE.COLKEYWORDNAMES.NUM_LINES: T
NORMAL: SOURCE.COLKEYWORDNAMES.SOURCE_ID: T
NORMAL: SOURCE.COLKEYWORDNAMES.SPECTRAL_WINDOW_ID: T
NORMAL: SOURCE.COLKEYWORDNAMES.TIME: T
NORMAL: SOURCE.COLKEYWORDNAMES.REST_FREQUENCY: T
NORMAL: SOURCE.COLVALUES.POSITION: T
NORMAL: SOURCE.COLVALUES.DIRECTION: T
NORMAL: SOURCE.COLVALUES.PROPER_MOTION: T
NORMAL: SOURCE.COLVALUES.CALIBRATION_GROUP: T
NORMAL: SOURCE.COLVALUES.CODE: T
NORMAL: SOURCE.COLVALUES.INTERVAL: T
NORMAL: SOURCE.COLVALUES.NAME: T
NORMAL: SOURCE.COLVALUES.NUM_LINES: T
NORMAL: SOURCE.COLVALUES.SOURCE_ID: T
NORMAL: SOURCE.COLVALUES.SPECTRAL_WINDOW_ID: T
NORMAL: SOURCE.COLVALUES.TIME: T
NORMAL: SOURCE.COLVALUES.REST_FREQUENCY: T
NORMAL: SOURCE.KEYWORDFIELDNAMES: T
NORMAL: DOPPLER.COLNAMES: T
NORMAL: DOPPLER.COLKEYWORDNAMES.DOPPLER_ID: T
NORMAL: DOPPLER.COLKEYWORDNAMES.SOURCE_ID: T
NORMAL: DOPPLER.COLKEYWORDNAMES.TRANSITION_ID: T
NORMAL: DOPPLER.COLKEYWORDNAMES.VELDEF: T
NORMAL: DOPPLER.COLVALUES.DOPPLER_ID: T
NORMAL: DOPPLER.COLVALUES.SOURCE_ID: T
NORMAL: DOPPLER.COLVALUES.TRANSITION_ID: T
NORMAL: DOPPLER.COLVALUES.VELDEF: T
NORMAL: DOPPLER.KEYWORDFIELDNAMES: T
NORMAL: SYSCAL.COLNAMES: T
NORMAL: SYSCAL.COLKEYWORDNAMES.ANTENNA_ID: T
NORMAL: SYSCAL.COLKEYWORDNAMES.FEED_ID: T
NORMAL: SYSCAL.COLKEYWORDNAMES.INTERVAL: T
NORMAL: SYSCAL.COLKEYWORDNAMES.SPECTRAL_WINDOW_ID: T
NORMAL: SYSCAL.COLKEYWORDNAMES.TIME: T
NORMAL: SYSCAL.COLKEYWORDNAMES.TSYS: T
NORMAL: SYSCAL.COLKEYWORDNAMES.TSYS_FLAG: T
NORMAL: SYSCAL.COLKEYWORDNAMES.BIMA_JYPERK: T
NORMAL: SYSCAL.COLVALUES.ANTENNA_ID: T
NORMAL: SYSCAL.COLVALUES.FEED_ID: T
NORMAL: SYSCAL.COLVALUES.INTERVAL: T
NORMAL: SYSCAL.COLVALUES.SPECTRAL_WINDOW_ID: T
NORMAL: SYSCAL.COLVALUES.TIME: T
NORMAL: SYSCAL.COLVALUES.TSYS: T
NORMAL: SYSCAL.COLVALUES.TSYS_FLAG: T
NORMAL: SYSCAL.COLVALUES.BIMA_JYPERK: T
NORMAL: SYSCAL.KEYWORDFIELDNAMES: T
```

1.1.3 miriadtoms - Function

Package bima
Module mirfiller

fill a measurement set from a Miriad dataset

Synopsis

miriadtoms(msfile, mirfile, defpass, verbose)

Description

This global function provides a one-step method for filling Miriad data in the most typical circumstances, such as with simple raw BIMA data or calibrated BIMA data. If one needs to take control of the data selection, create several measurement sets from a single miriad dataset, or set special loading options, one should create a mirfiller tool.

Arguments

msfile	the output measurement set name
mirfile	the input Miriad dataset name
defpass	a string indicating the window passing mode to use. This mode affects the which kinds of windows get selected. Allowed: 'default' – let mirfiller determine which mode to use based on what is found in the data; 'rawbima' – use a default appropriate for uncalibrated BIMA data: pass all spectral line windows and both side band averages but no window averages; 'calbima' – use a default appropriate for calibrated data: pass all spectral line windows but no wideband averages; 'all' – pass all windows and wideband channels. Default: 'default'
verbose	if true, send lots of messages to the logger. A summary of the input miriad dataset will be printed, and filling will be done in verbose mode. Default: F

1.2 bimafiller - Module

Module for converting a MIRIAD visibility file to a MeasurementSet

Tools

bimafiller	Tool for converting a MIRIAD visibility file to a MeasurementSet (deprecated)

Description

The `bimafiller` module containst the tool for converting a MIRIAD visibility file to an AIPS++ MeasurementSet.
NOTE: This module is **deprecated**. The `mirfiller` module replaces this one and should be used instead for converting MIRIAD visibility files.

1.2.1 bimafiller - Tool

Package	**bima**
Module	**bimafiller**

Tool for converting a MIRIAD visibility file to a MeasurementSet (deprecated)

Constructors

bimafiller	Construct the bimafiller tool

Functions

done	Destroy the bimafiller tool
fill	Fill the data
gui	Bring up a GUI for user interaction
setoptions	Set options to control the filling.

include "bimafiller.g"

Description

The `bimafiller` tool is used to convert a MIRIAD visibility file to an AIPS++ MeasurementSet.
NOTE: This tool is **deprecated**. The `mirfiller` tool replaces `bimafiller` and should be used instead for converting MIRIAD visibility files.

bimafiller - Constructor

Package	**bima**
Module	**bimafiller**
Tool	**bimafiller**

Construct the bimafiller tool

Synopsis
bimafiller(msname, mirname)

Arguments

msname	Filename for MeasurementSet to create Allowed:String Default: ”
mirname	Name of an input miriad visibility dataset (note, miriad datasets are directories). Allowed:String Default: ”

Returns
bimafiller tool

Description

This constructor just creates the bimafiller tool, it does not do any actual filling.

Example

```
- mybf := bimafiller(msname='3c273.ms', mirname='3c273.vis')
```

---

bimafiller.fill - Function

Package	**bima**
Module	**bimafiller**
Tool	**bimafiller**

Fill the data

Synopsis
fill()

Returns
T or F

Description

This function does the actual filling of the data from miriad format file to a MeasurementSet.

Example
Fill the data from the miriad visibility file `3c273.vis` into the MeasurementSet `3c273.ms`.

```
- mybf := bimafiller(msname='3c273.ms', mirname='3c273.vis')
- mybf.fill()
- mybf.done()
```

---

bimafiller.setoptions - Function

Package	**bima**
Module	**bimafiller**
Tool	**bimafiller**

Set options to control the filling.

Synopsis
setoptions(msname, mirname)

Arguments

msname	Filename for MeasurementSet to create Allowed:String Default: ”
mirname	Name of an input miriad visibility dataset Allowed:String Default: ”

Description

Set the options to control the filling of the MeasurementSet. Currently, only the input visibility file and output MeasurementSet can be chosen, but further options will soon be added.

Returns
T or F

Example

```
- mybf := bimafiller(mirname='3c273.vis')
- mybf.setoptions(msname='3c273.ms')
- mybf.fill()
- mybf.done()
```

---

bimafiller.gui - Function

Package	**bima**
Module	**bimafiller**
Tool	**bimafiller**

Bring up a GUI for user interaction

Synopsis
gui()

Returns
T or F

Description

This brings up a small GUI which allows the user to set the parameters for the filler. Currently, the only parameters that can be set are the name of the input visibility dataset and the output MeasurementSet. Logging of output from the filler is shown in the GUI as well. The filling can also be initiated from this tool.

Example

```
- mybf := bimafiller(mirname='3c273.vis', msname='3c273.ms')
- mybf.gui()
- mybf.done()
```

---

bimafiller.done - Function

Package	**bima**
Module	**bimafiller**
Tool	**bimafiller**

Destroy the bimafiller tool

Synopsis
done()

Returns
T or F

Description

This destroys the `bimafiller` tool.

1.3 bimams - Module

Module for BIMA measurement set access and manipulation
include "bimams.g"

Description

`bimams` provides an interface to BIMA-specific measurement sets. It provides several convenience functions for accessing information in measurement set subtables. `bimams` is a vertically integrated tool built on top of `ms` (another way of looking at it is that a `bimams` tool contains an `ms` tool). Thus, the usefulness of `bimams` is that it supplies a simple interface for observers to access and manipulate measurement set information that may be utilized more by BIMA than other telescopes.

Overview of `bimams` tool functions:

miscellaneous timerange

miscellaneous totalintegrationtime fieldids

spectral window chanwidths, startfreqs, endfreqs, namespw, nchans, nspw, reavg, reffreqs, spwid, spwidsbyname, spwids, spwnames, spwnames startfreqs

polarization polartable, polarids stokesid

table access ddtable ddid ddids fieldtable mcaddids sourcetable spwtable subtable

underlying tools ms

Tools

bimams	tool for manipulating BIMA measurement sets

1.3.1 bimams - Tool

Package bima
Module bimams

tool for manipulating BIMA measurement sets

Description
bimams is a tool which is built on top of an ms tool and provides easy access

to information frequently requested by BIMA users.

Constructors

bimams	Construct a bimams tool

Functions

chanwidths	Get the channel widths of the specified type of windows
ddid	get the DATA_DESCRIPTION ID (row number) associated with a specified spectral window id and a polarization or polarization id.
ddids	get the DATA_DESCRIPTION IDs (row numbers) associated with specified spectral window ids and polarization ids. The spectral window and polarization ids are or-ed within themselves and then anded together.
ddtable	get the DATA_DESCRIPTION table associated with the ms
doptable	get the DOPPLER table associated with the ms
endfreqs	get the frequencies of the last channels in the multi-channel data spectral windows.
fieldids	Get the 1-based field ids associated with the specified source
fieldtable	Get the FIELD subtable as a table tool
mcaddids	Given a list of 1-based DATA DESCRIPTION ids which describe multi-channel data spectral windows, get a list of 1-based DATA DESCRIPTION ids which correspond to the multi-channel average windows.
ms	get the ms tool contained within this bimams tool.
namespw	Assign a name to a spectral window.
namespwid	Assign a name to a spectral window by specifying its id
nchans	Get the number of channels in each multi-channel data spectral window
nspw	Get the number of spectral windows
polarids	Get a list of polarization ids
polartable	Get the POLARIZATION subtable
reavg	Recompute sideband averages and channel averages based on channel data flags
reffreqs	Get a list of spectral window reference frequencies
sourcetable	Get the SOURCE subtable as a table tool
spwid	Get a spectral window id
spwids	Get a list of spectral window ids by specifying a type of spectral window and, optionally, a sideband
spwidsbyname	Get a list of spectral window ids by specifying a list of spectral window names
spwnames	Get a list of spectral window names by specifying the type of spectral window
spwtable	Get the SPECTRAL_WINDOW subtable as a table tool
startfreqs	Get the frequencies of the first channels in the multi-channel data spectral windows.
stokesid	Get the system-defined id for the specified stokes parameter.
subtable	Get the specified subtable of the underlying measurement set
timerange	Get the maximum and minimum times in the MS
totalintegrationtime	Get the total integration time of this MS

bimams - Constructor

Package	**bima**
Module	**bimams**
Tool	**bimams**

Construct a bimams tool

Synopsis
bimams(msname)

Description
This is used to construct a `bimas` tool associated with a measurement set. A Glish variable containing the tool functions of bimams is returned.

Arguments

msname	Name of the measurement set on disk to associate with this bimams tool Allowed:Disk measurement set name

Returns
bimams tool

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('sgrb2n.ms');
```

---

bimams.chanwidths - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get the channel widths of the specified type of windows

Synopsis
chanwidths(type)

Description
Get a Glish vector containing the channel widths (in Hz) for the specified type of windows. The order of the values in the returned vector is the same as the order of the type of windows specified in the SPECTRAL_WINDOWS subtable. For the multichannel data, the assumption is made that the channel

width is constant within a given spectral window (note though that the measurement set specification permits varying channel widths within a window; in general however, channel widths within a spectral window are constant for BIMA data). The actual values which are returned are the widths of the first channels in each window.

Arguments

type	type of spectral window Allowed:'s' (sibeband average), 'c' (multi-channel data), or 'a' (multi-channel average) Default: 'c'

Returns

A Glish vector of doubles containing the channel widths of the multi-channel data spectral windows.

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# get the widths of the channels in the multi-channel data windows of
# the underlying ms
bms.chanwidths();
```

The result is [-48828.125 -3125000 -3125000 -48828.125 -3125000 -3125000 48828.125 3125000 3125000 48828.125 3125000 3125000]

---

bimams.ddid - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

get the DATA_DESCRIPTION ID (row number) associated with a specified spectral window id and a polarization or polarization id.

Synopsis

ddid(spwid, stokes, polid)

Description
Retrieve the (1-based) DATA_DESCRIPTION ID (row number) based on the SPECTRAL_WINDOW ID and stokes parameter or polarization id. Returns -1 if the specified description doesn't exist in the table.

Arguments

spwid	The 1-based id (row number) of the spectral window in the SPECTRAL_WINDOWS subtable Allowed:Integer
stokes	The stokes parameter associated with the data description. Not used if polid is specified Allowed:String representation of a valid stokes parameter. Default: YY
polid	The 1-based id (row number) of the desired polarization in the POLARIZATION subtable. If specified, stokes is ignored. Allowed:Integer > 0

Returns
Integer (-1 if requested id does not exist)

Example

```
# get a data description id based on a spectral window id and a stokes
# parameter
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
#  get the data description id associated with spectral window id 10
# a stokes parameter of YY
bms.ddid(spwid=10,stokes='YY');
```

The result is 10.

bimams.ddids - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

get the DATA_DESCRIPTION IDs (row numbers) associated with specified spectral window ids and polarization ids. The spectral window and polarization ids are or-ed within themselves and then anded together.

Synopsis
ddids(spwids, polids)

Description
Retrieve (1-based) DATA_DESCRIPTION IDs (row numbers) based on SPECTRAL_WINDOW IDs and polarization ids.

Arguments

spwids	The 1-based ids (row numbers) of the spectral windows in the SPECTRAL_WINDOWS subtable Allowed:Vector of integers Default: unset (all spectral windows)
polids	The 1-based ids (row numbers) of the desired polarizations in the POLARIZATION subtable. Allowed:Vector of integers Default: unset (all polarizations)

Returns
Vector of integers (empty vector of no specified ids are found)

Example

```
# get a list of data description ids
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
#  get the data description id associated with spectral window ids 5
# and 10 and polarizaion id 0
bms.ddid(spwids=[5,10],polids=0);
```

The result is [5,10].

---

bimams.ddtable - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

get the DATA_DESCRIPTION table associated with the ms

Synopsis
ddtable()

Description
Retrieve the DATA_DESCRIPTION subtable as a table tool.

Returns
Table tool

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# browse its DATA_DESCRIPTION subtable
bms.ddtable().browse();
```

---

bimams.doptable - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

get the DOPPLER table associated with the ms

Synopsis
doptable()

Description
Retrieve the DOPPLER subtable as a table tool.

Returns
Table tool

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# browse its DOPPLER subtable
bms.doptable().browse();
```

---

bimams.endfreqs - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

get the frequencies of the last channels in the multi-channel data spectral windows.

Synopsis
endfreqs()

Description
Retrieve a Glish vector of doubles representing the end (last channel) frequencies (in Hz) in the multi-channel spectral windows. The order of the frequencies is the same as the order of the multi-channel windows in the SPECTRAL_WINDOWS table.

Returns
Vector of Doubles

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# get the end frequencies of the multi-channel spectral windows
bms.endfreqs();
```

The result is `[9.3162917e+10 9.30201938e+10 9.29201938e+10 9.2862413e+10 9.27202078e+10 9.26202078e+10 9.61141944e+10 9.62569176e+10 9.63569176e+10 9.64146984e+10 9.65569036e+10 9.66569036e+10]`

---

bimams.fieldids - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get the 1-based field ids associated with the specified source

Synopsis
fieldids(sname)

Description
This function returns a vector of 1-based field ids associated with the specified source.

Arguments

sname	The source to which to get the field ids for Allowed:String Default: None

Returns
Vector of Ints

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
- bms := bimams('1733-130');
# the the 1-based field ids for the source 1733-130
- bms.fieldids('1733-130');
1
```

---

bimams.fieldtable - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get the FIELD subtable as a table tool

Synopsis
fieldtable()

Description
Get the FIELD subtable as a table tool.

Arguments

Returns
Table tool

Example

```
- # construct a bimams tool and assign it to a Glish variable named bms
- bms := bimams('1733-130');
- # get the FIELD table
- ft := bms.fieldtable();
- # browse it
- ft.browse();
```

---

bimams.mcaddids - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Given a list of 1-based DATA DESCRIPTION ids which describe multi-channel data spectral windows, get a list of 1-based DATA DESCRIPTION ids which correspond to the multi-channel average windows.

Synopsis
mcaddids(mcd)

Description
Given a list of 1-based DATA DESCRIPTION ids which describe multi-channel data spectral windows, get a list of 1-based DATA DESCRIPTION ids which correspond to the multi-channel average windows.

Arguments

mcd	vector of 1-based data description ids which refer to multichannel data windows. Allowed:Vector of integers

Returns
Vector of integers. If any input DATA DESCRIPTION ids do not have multi-channel average associations, -1 is in the corresponding the position in the returned vector

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# data description ids 1,3,5 are multichannel data.  1 and 3 have
# corresponding multichannel averages, while 5 does not.
bms.mcaddids([1,3,5]);
```

The result is [15,17,-1].

---

bimams.ms - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

get the ms tool contained within this bimams tool.

Synopsis
ms()

Description
Retrieve the `ms` tool contained within this `bimams` tool (recall that a `bimams`

tool is essentially a container for an `ms` tool which supplies convience functions for retrieving information from the underlying `ms` tool).

Returns
ms tool

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# get the contained ms
true_ms := bms.ms();
true_ms.name();
```

The result of the last command is `neptune.ms`.

---

bimams.namespw - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Assign a name to a spectral window.

Synopsis
namespw(sb, type, mcn, name)

Description
Assign a name to a spectral window (that is, insert a value into the NAME column of the specified window's row in the SPECTRAL_WINDOWS table). After assigning a name, other functions can be used to retrieve information on spectral windows by specifying their names. BIMA vertical integration tools require that user-specified spectral window names be unique (this is not a requirement of the measurement set specification, however). Therefore, this function will not assign a name to a spectral window if another window different from the specified one already has that name. In this case, this function will fail.

Arguments

sb	The sideband in which the window is located. Allowed:'u' (usb) or 'l' (lsb)
type	type of spectral window Allowed:'s' (sibeband average), 'c' (multi-channel data), or 'a' (multi-channel average)
mcn	the 1-based location of this window in the SPECTRAL_WINDOWS table with respect to other windows of the same type and sideband (usually NOT the same as its row number in this table). Not used if type='s' Allowed:Integer from 1 to the number of spectral windows of the specified type in the specified sideband Default: 1
name	the name to assign to this window Allowed:String

Returns
Boolean: T if naming was successful.

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# name the third multichannel data window in the lower sideband HCN
bms.namespw(sb='l', type='c', mcn=3, name='HCN');
# The result of the previous command is T if there is no window already
# named HCN. If there is, the function will fail.

# Now get the spectral window id of the window just named by using its
# name

bms.spwidsbyname('HCN');

# the result of the previous command is the spectral window id (the
# row number of the spectral window in the SPECTRAL_WINDOWS table. For
# example, 22.
```

---

bimams.namespwid - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Assign a name to a spectral window by specifying its id

Synopsis
namespwid(spwid, name)

Description
Assign a name to a spectral window (that is, insert a value into the NAME column of the specified window's row in the SPECTRAL_WINDOWS table) by specifying the window's id (ie, row number in the SPECTRAL_WINDOWS table). After assigning a name, other functions can be used to retrieve information on spectral windows by specifying their names. BIMA vertical integration tools require that user-specified spectral window names be unique (this is not a requirement of the measurement set specification, however). Therefore, this function will not assign a name to a spectral window if another window different from the specified one already has that name. In this case, this function will fail.

Arguments

spwid	The 1-based id of the spectral window (ie, the row number of the window in the SPECTRAL_WINDOWS table) Allowed:Integer >= 1 and <= number of spectral windows
name	the name to assign to this window Allowed:String

Returns
Boolean: T if naming was successful.

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# name window number 3 CS
bms.namespw(spwid=3, name='CS');
# The result of the previous command is T if there is no window already
# named CS. If there is, the function will fail.
```

---

bimams.nchans - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get the number of channels in each multi-channel data spectral window

Synopsis
nchans()

Description
Get a Glish vector of integers of the number of channels in the multi-channel data spectral windows. The order of the values is the same as that of the multi-channel data windows in the SPECTRAL_WINDOWS subtable.

Returns
vector of integers of the number of channels for the multi-channel data spectral windows

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# get the number of channels in the multi-channel data spectral
# windows of this measurement set
bimams.nchans();
```

The result of the last command will be something like `[256 32 32 256 32 32 256 32 32 256 32 32]`

---

bimams.nspw - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get the number of spectral windows

Synopsis
nspw()

Description
Get the number of spectral windows in the SPECTRAL_WINDOWS subtable. This includes all tpes of spectral windows; that is, the returned value is simply the number of rows in this subtable

Returns
Integer: the number of rows in the SPECTRAL_WINDOWS subtable

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# get the number of spectral windows in this measurement set
bimams.nspw();
```

The result of the last command will be something like `26`.

---

bimams.polartable - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get the POLARIZATION subtable

Synopsis
polartable()

Description
Get the POLARIZATION subtable associated with the underlying measurement set. The returned quantity is a table tool.

Returns
Table tool containing the POLARIZATION subtable

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# browse the POLARIZATION subtable associated with the underlying
# measurement set
bimams.polartable().browse();
```

---

bimams.polarids - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get a list of polarization ids

Synopsis
polarids(stokes)

Description
Get a 1-based vector of integers representing the polarization ids (row numbers) from the POLARIZATION subtable by specifying a stokes parameter. If no such polarization exists in the subtable, an empty vector is returned.

Arguments

stokes	The stokes parameter associated with the polarization id (e.g., 'YY', 'I') Allowed:String

Returns
Vector of integers: the corresponding polarization ids or an empty vector if the specified polarization is not found

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# get the polarization id associated with the YY stokes parameter
bimams.polarid('YY');
```

The last command will return the corresponding row in the POLARIZATION subtable, for example `1`

---

bimams.reavg - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Recompute sideband averages and channel averages based on channel data flags

Synopsis
reavg(out, dosort, reset, verbosity)

Description
This function recomputes the sideband averages and channel averages based on the channel data flags. It is currently implemented in glish and is rather slow. The plan is to implement it in C++ so it will run faster. In general, it should be run after channel data have been flagged. It is meant to be the AIPS++ analog of MIRIAD's `uvwide` task.

Yet to do: `edge` and `blankf` parameters, as in `uvwide` which allow the user to discard edge channels when recomputing averages, still must be added. Note that in `uvwide`, the default value of `blankf` is 0.033 which is why `uvwide` run with this default will compute different averages than the current implementation of `reavg`.

Arguments

out	the output dataset, if not specified the input dataset is modified and no new dataset is written Allowed:String Default: ”
dosort	sort the data when doing subqueries, if F, will speed up execution, but should only be set to F if you are sure the dataset is already sorted in time-data description id-baseline order (the order in which mirfiller writes data when filling) Allowed:Boolean Default: T
reset	should all average data be recomputed? By default (reset=T), all averages recomputed regardless of the value of the existing flags, otherwise (reset=F), only averages with F flags are recomputed (i.e. averages flagged bad are left untouched). Note that setting reset=F will not increase the speed of the function. Allowed:Boolean Default: T
verbosity	Controls the level of informational messages, 0 = run quietly, higher numbers = more messages Allowed:Integer Default: 1

Returns
T or fail

Example

```
# flag some data
- af := autoflag('neptune.ms');
- af.setselect(clip=[expr='ABS YY',min=0.001,max=50]);
# construct a bimams tool and assign it to a Glish variable named bms
- bms := bimams('neptune.ms');
# recompute sideband averages and print lots of messages during execution
- bms.reavg('neptune.ms.reavg',verbosity=3);
T
```

---

bimams.reffreqs - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get a list of spectral window reference frequencies

Synopsis
reffreqs()

Description
Get a vector of doubles representing the reference frequencies (in Hz) of all the spectral windows in the SPECTRAL_WINDOWS table in the order they appear in this table.

Returns
Vector of doubles: the spectral window reference frequencies (in Hz)

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# get the reference frequencies of the spectral windows from the
# underlying measurement set
bimams.reffreqs;
```

The last command will return the reference frequencies (in Hz), e.g.,
[9.31753682e+10 9.31170688e+10 9.30170688e+10 9.28748642e+10
9.28170828e+10 9.27170828e+10 9.61017432e+10 9.61600426e+10
9.62600426e+10 9.64022472e+10 9.64600286e+10 9.65600286e+10]

---

bimams.sourcetable - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get the SOURCE subtable as a table tool

Synopsis
sourcetable()

Description

Get the SOURCE subtable as a table tool.

Arguments

Returns
Table tool

Example

```
- # construct a bimams tool and assign it to a Glish variable named bms
- bms := bimams('1733-130');
- # get the SOURCE table
- st := bms.sourcetable();
- # browse it
- st.browse();
```

---

bimams.spwid - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get a spectral window id

Synopsis
spwid(sb, type, mcn)

Description
Get a 1-based spectral window id (row number) from the SPECTRAL_WINDOW subtable by specifying a sideband, spectral window type, and the position of the desired spectral window relative to other windows of the same type in the same sideband. If no window matching the specified parameters is found, -1 is returned.

Arguments

sb	The sideband in which the desired window is located Allowed:String: 'l' (lsb) or 'u' (usb)
type	The type of window Allowed:String: 's' (sideband average), 'c' (multi-channel data), or 'a' (multi-channel average)
mcn	The 1-based position of the desired window with respect to other windows of the same type in the same sideband. Usually NOT the same as the row number in the SPECTRAL_WINDOWS table Allowed:Integer: >= 1 and <= the total number of windows of the specified type in the specified sideband. Not used if type='s' Default: 1

Returns
Integer: the id of the specified window or -1 if not found

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# get the spectral window id of the the third multi-channel average
# window in the upper sideband
bms.spwid(sb='u',type='a',mcn=3);
```

The last command will return the corresponding row in the SPECTRAL_WINDOWS subtable, for example `7` or `-1` if the specified window does not exist.

---

bimams.spwidsbyname - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get a list of spectral window ids by specifying a list of spectral window names

Synopsis
spwidsbyname(names)

Description
Get a 1-based vector of spectral window ids (row numbers) from the SPECTRAL_WINDOW subtable by specifying a vector of names of the windows. A vector of integers of the same length as the input name vector is returned. For names which are not found in the spectral window table, `-1` will occupy that place in the returned vector.

Arguments

names	The names of the spectral windows Allowed:Vector of strings

Returns
Vector of integers: the ids (row numbers) of the specified windows (-1 for names which do not exist in the table)
.

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# assign spectral window number 3 the name of HCN
bms.namespwid(id=3,name='HCN');
# assign spectral window number 7 the name of CS
bms.namespwid(id=7,name='CS');
# get the spectral window ids of the window named CS, HCN, and SiO
bms.spwidsbyname(names = ['CS','HCN','SiO']);
```

The last command returns `[7, 3, -1]` (the user has not yet named a window SiO, which is why the final value in the vector is `-1`).

---

bimams.spwids - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get a list of spectral window ids by specifying a type of spectral window and, optionally, a sideband

Synopsis

spwids(type, sb)

Description

Get a 1-based vector of spectral window ids (row numbers) from the SPECTRAL_WINDOW subtable by specifying a type of spectral window (sideband average, multi-channel data, multi-channel average) and, optionally, a sideband. A vector of integers of the ids of all windows of the specified type in the specified sideband is returned.

Arguments

type	The spectral window type Allowed: String: 'A' (all types), 'a' (multi-channel average), 'c' (multi-channel data), or 's' (sideband average)
sb	The sideband. Not used if type='s' Allowed: String: 'b' (both), 'l' (lsb), 'u' (usb) Default: 'b'

Returns

Vector of integers: the ids (row numbers) of all the windows of the specified type.

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# get the a list of ids of all the multi-channel data windows in both
# sidebands
bms.spwids(type='c', sb='b');
# The previous command returns a vector of integers representing the
# ids of all the multi-channel data windows, e.g., [4 6 8 10 12 14 16 18
# 20 22 24 26].

# get the ids for the multi-channel data windows from the upper
# sideband only
bms.spwids(type='c', sb='u');
# the previous command returns [16 18 20 22 24 26]
```

bimams.spwnames - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get a list of spectral window names by specifying the type of spectral window

Synopsis
spwnames(type)

Description
Get a vector of strings representing the names of all the spectral windows of a specified type.

Arguments

type		The type of spectral window
	Allowed:	String: 'all' (all types), 'a' (multi-channel averages), 'c' (multi-channel data), 's' (sideband averages)
	Default:	'all'

Returns
Vector of strings: the names of the spectral windows of the specified type in the order the windows appear in the SPECTRAL_WINDOWS subtable

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# get the names of the multi-channel data windows
bms.spwnames(type='c');
```

The last command returns a vector of the names of the multi-channel data windows, e.g., `none none HCN none none CS none none SiO none none none`. `none` is the default entry in the NAME column.

---

bimams.spwtable - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get the SPECTRAL_WINDOW subtable as a table tool

Synopsis
spwtable()

Description
Get the SPECTRAL_WINDOW subtable as a table tool.

Arguments

Returns
Table tool

Example

```
- # construct a bimams tool and assign it to a Glish variable named bms
- bms := bimams('1733-130');
- # get the SPECTRAL_WINDOW table
- spwt := bms.spwtable();
- # browse it
- spwt.browse();
```

---

bimams.startfreqs - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get the frequencies of the first channels in the multi-channel data spectral windows.

Synopsis
startfreqs()

Description

Retrieve a Glish vector of doubles representing the start (first channel) frequencies (in Hz) in the multi-channel spectral windows. The order of the frequencies is the same as the order of the multi-channel windows in the SPECTRAL_WINDOWS table.

Returns
Vector of doubles of start frequencies (in Hz) for the multi-channel data spectral windows

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# get the starting frequencies of the multi-channel spectral windows
bms.startfreqs();
```

The result is a vector of doubles representing the start frequencies for the multi-channel data windows, e.g., `[9.31753682e+10 9.31170688e+10 9.30170688e+10 9.28748642e+10 9.28170828e+10 9.27170828e+10 9.61017432e+10 9.61600426e+10 9.62600426e+10 9.64022472e+10 9.64600286e+10 9.65600286e+10]`.

---

bimams.stokesid - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get the system-defined id for the specified stokes parameter.

Synopsis
stokesid(stokes)

Description
Get the system-defined id for the specified stokes parameter as defined in `aips/implement/Measures/Stokes.h`.

Arguments

stokes	The stokes parameter (e.g., 'YY', 'I', etc.) Allowed:String: a stokes parameter

Returns
Integer: The system defined stokes id (see
`aips/implement/Measures/Stokes.h`
`) or -1 if the stokes parameter is not recognized`

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# get the stokes id for stokes parameter YY
bms.stokesid('YY');
```

The last command returns 12.

---

bimams.subtable - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get the specified subtable of the underlying measurement set

Synopsis
subtable(tname, readonly)

Description
Retrieve the specified subtable from the underlying measurement set as a table tool. By default, the table is opened in read-only mode, but it can be opened in read-write mode by setting the readonly parameter to F.

Arguments

tname	name of the subtable Allowed:String
readonly	open the table in read-only mode (no writing allowed) Allowed:Boolean Default: T

Returns
Table tool containing the specified subtable

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
bms := bimams('neptune.ms');
# browse the WEATHER table
bms.subtable('WEATHER').browse();
```

---

bimams.timerange - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get the maximum and minimum times in the MS

Synopsis
timerange()

Description
This function returns the maximum and minimum times in the MS. The thing returned is a record with fields `max` and `min`. Each field contains a quantity corresponding to the maximum and minimum times, respectively.

Arguments

Returns
A record containing the maximum and minimum times
.

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
- bms := bimams('1733-130');
# get the maximum and minimum times
- tra := bms.timerange();
- tra
[max=[value=4.38575597e+09, unit=s], min=[value=4.38573665e+09, unit=s]]
# now convert the maximum time to something meaningful to a human
- dq.time(tra.max,form='ymd');
1997/11/09/01:32:52.170
```

---

bimams.totalintegrationtime - Function

Package	**bima**
Module	**bimams**
Tool	**bimams**

Get the total integration time of this MS

Synopsis
totalintegrationtime(gaptime)

Description
This function returns the total integration time in the MS.

Arguments

gaptime	The minimum time between scans, if <0, 2 times the maximum value in the INTERVAL column is used Allowed:Double Default: -1

Returns
Quantity
.

Example

```
# construct a bimams tool and assign it to a Glish variable named bms
- bms := bimams('1733-130');
```

```
# get the total integration time
- bms.totalintegrationtime();
[value=2044.13995, unit=s]
```

1.4 bimacalibrater - Module

Module for calibrating BIMA data
include "bimacalibrater.g"

Description

The `bimacalibrater` module provides functions for calibrating BIMA data. The facilities of the `bimacalibrater` module are made available in Glish by issuing the following command:

```
- include 'bimacalibrater.g'
T
```

where a hyphen precedes user input. The Glish response is indicated without the prompt.
A `bimacalibrater` tool is created and attached to a specified set of measurement sets as indicated in the following example:

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],'1733-130.ms',
+ targetnames=['sgrb','sgra'],phcalnames='1733')
```

Many functions of `bimacalibrater` are wrappers around corresponding functions in `calibrater`. One important exception is `transfer()` which transfers the solutions from one calibration table (usually the one written by `solve()`) to another table (generally the one that will be applied to the target source with `correct()`). This is generally a necessary step because, for BIMA data, the calibration solutions are determined for a small set of spectral windows (e.g., the two sideband averages [the wideband data in miriad parlence]), but these solutions must be applied to all the windows in the target datasets. The only way to do this in AIPS++ is to create a new calibration table which reflects the number of spectral windows in the target sources.
Essentially, a `bimacalibrater` tool creates (on demand) and stores `calibrater` tools attached to the Measurement Sets specified in the `bimacalibrater` tool constructor. This tool's various functions manipulate these `calibrater` tools. In general, the functions take a parameter called `sources` which tells the function which `calibrater` tools it should manipulate. The `sources` parameter takes a vector of strings. The strings are source nicknames (see below) and/or source roles. Recognized roles are `'all'` (all sources), `'targets'` (target sources), `'phcals'` (phase calibrators),

'pbcals' (passband calibrators), and 'fcals' (flux calibrators). The source roles are specified by the user when the tool is constructed (see below).
Here is an example of what a typical calibration session using bimacalibrater might look like:

```
bc := bimacalibrater('sgrb2.ms','1733-130.ms',targetnames='narrow',
   phcalnames='wide');
bc.setjy(fluxdensity=[5,0,0,0]);
bc.setsolve(type='G',refant=4,table='gcal',t=600);
bc.solve();
bc.fit();
bc.transfer('gcal.xfer','gcal');
bc.setapply();
bc.correct();
```

Here, a bimacalibrater tool is constructed by specifying a list of target sources ('sgrb2'), nicknames for these sources ('narrow'), a list of phase calibrators ('1733-130.ms'), and a list of nicknames for these calibrators ('wide'). The nicknames are optional and are meant to provide an easy way of referring to datasets (sometimes the names of measurement sets can be long, especially when fully qualified paths must be specified). The nicknames need not be provided, in which case they are set to the names of the corresponding measurement sets.
The flux density scale of the calibrater is set with setjy (which is actually a wrapper of the imager function of the same name). This function will optinally look up the flux density of the calibrator from the BIMA calibrator flux density table and write the flux desnity nearest in time to the observations to the measurement set. Next, the solver is set up using setsolve() and solutions are generated using solve(). Both are wrappers around calibrater functions of the same names.
The fit() function launches the gainpolyfitter GUI which allows one to interpolate the solutions by fitting polynomials to them.
The transfer() function is the reason bimacalibrater exists. The usual procedure of calibrating BIMA data is to generate solutions for the sideband average data of the calibrator. Solutions from these two windows are then applied to all the windows of target source. However, in AIPS++ the number of windows in the calibration table must be the same as the number in the dataset to which it is being applied. Therefore, the solutions in the gcal table, which contains only two windows, must be transferred to a new table gcal.xfer which has the same number of windows as the target source dataset (e.g. 10 [2 sideband average windows + 8 channel data windows]). Once this is done, further information on how to apply the solutions can be provided with setapply() and finally the solutions can be applied with correct().

Tools

bimacalfluxtable	Tool to retrieve information from the BIMA calibrator tables
bimacalibrater	Calibration of BIMA data

Functions

bimacalibratertest	Test bimacalibrater

1.4.1 bimacalibrater - Tool

Package	**bima**
Module	**bimacalibrater**

Calibration of BIMA data

Description

A `bimacalibrater` tool performs all the necessary steps for calibrating BIMA data. It is meant to be a "track-based" tool; that is, it can hold all the datasets observed in a track and use them appropriately in various calibration steps.

Constructors

bimacalibrater	Construct a bimacalibrater tool from a set of MeasurementSets

Functions

addtargets	Add target sources
close	Close the calibrater tools associated with the specified measurement sets.
correct	Apply calibration
done	Done with this tool
fit	Interpolate gain solutions by fitting polynomials
getsourcenames	get a list of source nicknames
plotcal	Plot a calibration table(s)
setapply	Set how solutions should be applied
setdata	Set the data parameters selection for subsequent processing
setjy	Set the flux density of the specified sources
setsolve	Set up what calibration component is to be solved for and how the solution should be determined.
solve	Solve for the selected calibration components
summary	Print a summary of this tool.
transfer	Transfer calibration solutions from one table to another.
type	Get the type of this tool, namely 'bimacalibrater'

bimacalibrater - Constructor

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Construct a bimacalibrater tool from a set of MeasurementSets

Synopsis
bimacalibrater(targets, phcals, pbcals, fcals, targetnames, phcalnames, fcalnames, pbcalnames)

Description
Create a `bimacalibrater` tool.

Arguments

targets	Target source measurement set names Allowed:Vector of strings Default: ' '
phcals	Phase calibrator measurement set names Allowed:Vector of strings Default: ' '
pbcals	Passband calibrator measurement set names Allowed:Vector of strings Default: ' '
fcals	Flux density calibrator measurement set names Allowed:Vector of strings Default: ' '
targetnames	Target source nicknames Allowed:Vector of strings Default: unset (will set to targets)
phcalnames	Phase calibrator nicknames Allowed:Vector of strings Default: unset (will set to phcals)
fcalnames	Flux density calibrator nicknames Allowed:Vector of strings Default: unset (will set to fcals)
pbcalnames	Passband calibrator nicknames Allowed:Vector of strings Default: unset (will set to pbcals)

Returns
bimacalibrater tool

Example

```
# the '+' is printed by glish, not typed by you
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
```

---

bimacalibrater.getsourcenames - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

get a list of source nicknames

Synopsis
getsourcenames(roles)

Description

The `getsourcenames` function returns a list of source nicknames associated with the specified roles. The `roles` parameter should be a vector of strings, recognized values being `'all'`, `'targets'`, `'phcals'`, `'pbcals'`, `'fcals'` for all sources, target sources, phase calibrators, passband calibrators, and flux density calibrators, respectively. This function is meant to help users remember how they have named their sources (it is also used internally by other functions of `bimacalibrater`).

Arguments

roles		Return nicknames of sources of these roles
	Allowed:	Vector of strings. Recognized values are 'all' (all nicknames), 'targets' (target source nicknames), 'phcals' (phase calibrators), 'pbcals' (passband calibrators), and 'fcals' (flux density calibrators)
	Default:	none

Returns
array of corresponding nicknames or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.getsourcenames('all')
sgrb sgra 1733
- bc.getsourcenames(['phcals','targets'])
1733 sgrb sgra
- bc.getsourcenames('targets')
sgrb sgra
- bc.getsourcenames('phcals')
1733
```

---

bimacalibrater.setjy - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Set the flux density of the specified sources

Synopsis
setjy(sources, fieldid, spwid, fluxdensity)

Description

The `setjy` function is a wrapper around the `imager` tool's setjy function. It sets the flux density (by modifying the MODEL column) of the specified measurement sets. A list of nicknames (as well the the special groupings allowed in `getsourcenames`) can be given. If the specified flux density is <0, this function will attempt to find the calibrator in the BIMA calibrator flux density table and if successful, will use the flux density nearest in time (no attempt to adjust the flux density for frequency is currently made, nor is any interpolation in time attempted). In this case, there can only be a single source per measurement set. In the case where the user sets the flux density explicitly by passing a > 0 value to this function, only single model can be specified (it will be used for all the specified sources). Thus, you must run this function multiple times if you have calibrators of different flux densities when running in this mode.

Arguments

sources	Nicknames of sources to which to set the flux densities Allowed: Vector of strings (nicknames or roles of sources) Default: 'phcals'
fieldid	Field Id (1-relative) Allowed: Int Default: -1 (all)
spwid	Spectral Window Id. (1-relative) Allowed: Int Default: -1 (all)
fluxdensity	Specified flux density (I,Q,U,V) in Jy, if <0, attempt to look up flux density in BIMA calibrator flux density table Allowed: Vector of Doubles Default: -1

Returns
T or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- # all of these are equivalent for setting the I flux density to 5 Jy.
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setjy('1733',fluxdensity=[5,0,0,0]);
T
- bc.setjy('phcals',fluxdensity=[5,0,0,0]);
T
- bc.setjy(spwid=-1,fluxdensity=[5,0,0,0]);
T
# here we allow a lookup in the BIMA calibrator flux density table
- bc.setjy();
NORMAL: Using 2.79999995 Jy at 86.1999969 GHz measured on 1998/01/04/00:00:00.000
```

---

bimacalibrater.setdata - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Set the data parameters selection for subsequent processing

Synopsis
setdata(sources, mode, nchan, start, step, mstart, mstep, uvrange, sourcenames, msselect)

Description
This function is a wrapper around the `calibrater` tool's `sedata` function. The difference being that multiple sources can be specified, although the specified data selection will be the same for all of them. The `sources` parameter takes a list of source nicknames and/or a list of roles (acceptable roles are `'all','targets','phcals','pbcals','fcals'`; see `getsourcenames` for details). The `sourcenames` parameter allows selection of sources in multisource datasets. These names are read from the SOURCE subtable of the Measurement Set and the appropriate FIELD_ID selection is appended to the `msselect` string.

Arguments

sources	Nicknames of sources on which to run calibrater.setdata() Allowed: Vector of strings (nicknames or roles of sources) Default: 'phcals'
mode	Type of processing: channel or velocity Allowed: 'channel'|'velocity'
nchan	Number of channels to select Allowed: Int Default: 1
start	Start channel (1-relative) Allowed: Int
step	Step in channel number Allowed: Int
mstart	Start velocity (e.g. '20Km/s') Allowed: Quantity
mstep	Step in velocity (e.g. '100m/s' Allowed: Quantity
uvrange	UV range (in klambda) (e.g. [0, 50]) Allowed: Vector of Doubles Default: 0 (all data)
sourcenames	List of sources to select (only useful for multisource datasets; see above Allowed: Vector of strings Default: ' '
msselect	TAQL selection string Allowed: String

Returns
T or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel',uvrange=[10,50]);
T
```

---

bimacalibrater.setsolve - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Set up what calibration component is to be solved for and how the solution should be determined.

Synopsis

setsolve(sources, type, t, preavg, phaseonly, refant, table, append)

Description

The `setsolve` function is a wrapper around the `calibrater` tool's `setsolve` function. It allows the user to specify what calibration component is to be solved for as well as how the solution should be determined.

If the default value for the solution is specified, the following algorithm is used. The table is given the name `<solution type>cal` (e.g. `gcal` for `G` solutions). If this table already exists, then a number is appended (e.g., `gcal.2`). The table name is stored internally in the tool for use with other functions.

Arguments

sources	Nicknames of sources on which to run calibrater.setsolve() Allowed:Vector of strings (nicknames or roles of sources) Default: 'phcals'
type	Component type Allowed:"G","T","B","D" Default: "
t	Solution interval (in sec) Allowed:Float Default: 0.0
preavg	Pre-averaging interval (in sec) Allowed:Float Default: 0.0
phaseonly	Phase only solution ? Allowed:Bool Default: False
refant	Reference antenna Allowed:Int Default: -1
table	Output calibration table Allowed:Table name Default: " (see above for default naming scheme)
append	Append flag (set to T if you have multiple calibraters and want solutions in a single table) Allowed:Bool Default: False

Returns
Bool or fail

Example

```
# a single calibrater
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
```

```
# for two calibraters, putting all solutions in a single table
- bc := bimacalibrater('g34.3.ms',['1751+096.ms','2148+069.ms'],
  targetnames='src',phcalnames=['cal1','cal2']);
T
- bc.setjy(fluxdensity=[5,0,0,0]);
T
# solutions for both calibraters will go in table gcal2
- bc.setsolve(type='G',refant=4,table='gcal2',t=600,append=T);
T
```

---

bimacalibrater.solve - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Solve for the selected calibration components

Synopsis
solve(sources)

Description

The `solve` function wraps the `calibrater` tool's `solve` function. Solutions are generated and a calibration table is written for the specified sources. In addition, a keyword called `SPW_INFO` is written to the calibration table. This keyword contains information on the spectral window configuration of the source being solved for. This information is then read by the `transfer` function to help it make an educated guess about how the windows in the calibration written by `solve` map to the spectral windows in the sources to which the calibration will be applied.

Arguments

sources	Nicknames of sources on which to run calibrater.solve() Allowed: Vector of strings (nicknames or roles of sources) Default: 'phcals'

Returns
Bool or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- # bc.solve('phcals') or bc.solve('1733') are equivalent to the following
- bc.solve()
T
```

---

bimacalibrater.fit - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Interpolate gain solutions by fitting polynomials

Synopsis
fit(table)

Description

The `fit()` function creates a `gainpolyfitter` tool which allows the user to interpolate calibration solutions by fitting polynomials to them. This function lauches the GUI and then returns, which means you must be sure to complete the fits you want before proceeding to the next calibration step.

Arguments

table	The calibration table for which solutions are to be interpolated Allowed:Table name Default: unset (uses the table specified in setsolve())

Returns
T or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- bc.solve()
T
- bc.fit()
T
```

---

bimacalibrater.transfer - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Transfer calibration solutions from one table to another.

Synopsis
transfer(outtable, intable, spwmap, calibratees, forcecopy)

Description

The `transfer()` function is the reason `bimacalibrater` exists. It transfers solutions from one calibration table with `m` spectral windows to another table with `n` spectral windows (in general, `m != n`).
Here's how the function determines how to map the windows in the input table to those in the output table. It first checks the (optional) user specified `spwmap` parameter. With this parameter, the user describes the window mapping. For example, assume the input table has two windows representing solutions for the lower and upper sidebands (windows 1 and 2). Assume the target source has the following spectral window configuration:

```
window    description
---------------------------------
  1       lower sidebnad average
  2       upper sideband average
```

```
3-6      lower sideband channel data
7-10     upper sideband channel data
```

In order to calibrate these data correctly, the user would specify `spwmap=[1,2,1,1,1,1,2,2,2,2]`. Note that for this example, `transfer` is smart enough to determine this mapping without having the `spwmap` parameter specified (see below). Furthermore, the user can opt not to have solutions for some or all windows copied by specifying a value of `0` in the correct window slot of `spwmap`. If `spwmap` is not specified, `transfer` uses the following algorithm. If there is only a single spectral window in the input table, its solutions are copied to all windows of the output. If there are two windows, `transfer` assumes they represent lower and upper sideband solutions. It tries to determine which sideband corresponds to which window by reading the `SPW_INFO` keyword written by the `solve()` function of `bimacalibrater`. If it cannot find this information (e.g. if the input calibration table was written by `calibrater.solve()`), it assumes the first window contains the lower sideband solutions and the second contains the upper sidenbad solutions. It then reads the `SPECTRAL_WINDOW` table of the dataset to be corrected to determine in which sidebands its windows lie. For three or more spectral windows in the input calibration table with `spwmap` unspecified, this function will fail since there is no normal way to transfer solutions in this case. It is up to the user in this case to specify how the solutions are to be mapped.

Although effort has been made to try to make the best guesses possible, the best way to make sure you are getting what you want is to specify `spwmap`.

If the default value of `outtable` is specified, the output table name will be called `<intable>.xfer` (e.g., `gcal.xfer` if the input table is named `gcal`.

Arguments

outtable	Name of the calibration table to which to write solutions Allowed: Any valid table name Default: ''
intable	Name of the calibration table from which to transfer solutions Allowed: Any valid table name Default: '' (will use the table written by solve)
spwmap	Array describing the spectral window mapping from the input to the output (see above). The spectral windows are 1-based, and a value of 0 means not to transfer solutions for the specified window (GAINS are filled in with 0's) Allowed: Vector of Ints >= 0 Default: unset (transfer will do its best to guess how this should be done; see above)
calibratees	Nicknames or roles of sources to which the solutions will be applied (if more than one source is specified they all should have the same spectral window configuration) Allowed: Vector of strings Default: 'targets'
forcecopy	Force a copy of the input to output table when the input could be used to apply solutions Allowed: Boolean Default: T

Returns
T or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- bc.solve()
T
- bc.transfer('gcal.xfer','gcal',[1,2,1,1,1,1,2,2,2,2],'sgrb');
T
```

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Set how solutions should be applied

Synopsis
setapply(sources, type, t, table, select)

Description

The `setapply` function is a wrapper around the `calibrater` tool's `setapply` function. It is used to set calibration component information for applied calibration. Each calibration component represents a separate calibration matrix correction included in the measurement equation.

Arguments

sources	Nicknames of sources on which to run calibrater.setapply() Allowed: Vector of strings (nicknames or roles of sources) Default: 'targets'
type	Component type Allowed: "G","T","B","D" Default: 'G'
t	Interpolation interval (in sec) Allowed: Float Default: 0.0
table	Input calibration table Allowed: String Default: unset (will use table written by transfer(), or if transfer() hasn't been run, the table written by solve()
select	TAQL selection string Allowed: String Default: " "

Returns
Boolean or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- bc.solve()
T
- bc.transfer('gcal.xfer','gcal',[1,2,1,1,1,1,2,2,2,2],'sgrb');
T
- # bc.solve() is equivalent to the next line for this example
- bc.solve('sgrb','G',0,'gcal.xfer');
T
```

---

bimacalibrater.correct - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Apply calibration

Synopsis

correct(sources)

Description

This function is a wrapper around the `calibrater` tool's `correct()` function. It applies existing calibration data to the specified measurement sets, with initial table names and interpolation intervals set by the `setapply` function.

Arguments

sources	Nicknames of sources on which to run calibrater.correct() Allowed: Vector of strings (nicknames or roles of sources) Default: 'targets'

Returns
Boolean or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- bc.solve()
T
- bc.transfer('gcal.xfer','gcal',[1,2,1,1,1,1,2,2,2,2],'sgrb');
T
- bc.solve('sgrb','G',0,'gcal.xfer');
T
# bc.correct() is equivalent to the next line
- bc.correct('sgrb');
T
```

---

bimacalibrater.type - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Get the type of this tool, namely 'bimacalibrater'

Synopsis
type()

Description

This function simply returns the string `'bimacalibrater'`.

Returns
'bimacalibrater'

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.type();
bimacalibrater
```

---

bimacalibrater.addtargets - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Add target sources

Synopsis
addtargets(mss, names)

Description

This function adds target sources to the list of targets.

Arguments

mss	Names of the target MSs Allowed: Vector of strings Default: None
names	Nicknames for the target MSs Allowed: Vector of strings Default: unset (will use the values supplied in the mss parameter

Returns
Boolean or fail

Example

```
- bc := bimacalibrater([,,'1733-130.ms',phcalnames='1733');
- bc.setjy();
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- bc.solve()
T
- bc.addtargets('sgrb2.ms','sgrb2');
T
```

---

bimacalibrater.done - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Done with this tool

Synopsis
done()

Description

Release resources used by this tool.

Returns
Boolean T

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.done();
T
```

---

bimacalibrater.close - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Close the calibrater tools associated with the specified measurement sets.

Synopsis
close(sources)

Description

This function is a wrapper around the `calibrater` tool's `close()` function. It closes the calibrater tools (if they exist) associated with the specifed measurement sets.

Arguments

sources	Nicknames of sources on which to run calibrater.close() Allowed: Vector of strings (nicknames or roles of sources) Default: none

Returns
Boolean T or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- bc.solve()
T
- bc.transfer('gcal.xfer','gcal',[1,2,1,1,1,1,2,2,2,2],'sgrb');
```

```
T
- bc.solve('sgrb','G',0,'gcal.xfer');
T
- bc.correct('sgrb');
T
- bc.close('sgrb');
T
```

---

bimacalibrater.plotcal - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Plot a calibration table(s)

Synopsis
plotcal(sources, plottype, tablename, antennas, fields, polarization, spwids, timeslot, multiplot, nx, ny, psfile)

Description

This function is a wrapper around the `calibrater` tool's `plotcal()` function which plots calibration tables. `calibrater.plotcal()` is run seperately on each source specified by the `sources` parameter.
The following plottype choices are currently available 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 following plottype choices are currently available for D solutions:

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.

Arguments

sources	Nicknames of sources on which to run calibrater.plotcal() Allowed: Vector of strings (nicknames or roles of sources) Default: none
plottype	Plot type Allowed: String Default: 'AMP'
tablename	Calibration table name Allowed: String Default: ''
antennas	Antennas to plot Allowed: Vector of Ints Default: False
fields	Fields to plot Allowed: Vector of Ints Default: False
polarization	Polarization to plot Allowed: Integer Default: 1
spwids	Spectral windows id.'s to plot Allowed: Vector of Ints Default: False
timeslot	Timeslot index for B Jones Allowed: Integer Default: 1
multiplot	Turn on separate antenna plots Allowed: Bool Default: F
nx	If multiplot=T, number of plots on horizontal axis Allowed: Integer Default: 1
ny	If multiplot=T, number of plots on vertical axis Allowed: Integer Default: 1
psfile	name of output PostScript file to write plot to Allowed: String Default: empty string, causing plot to be sent to the screen

Returns
Boolean or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- bc.solve()
T
- bc.plotcal('1733',plottype='phase',table='gcal',antennas=[1,3]);
T
- bc.close('1733');
T
```

---

bimacalibrater.summary - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Print a summary of this tool.

Synopsis
summary()

Description

This function prints a summary of the current state of the tool.

Returns
Boolean T

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.summary()
```

```
<lots of state information gets printed>
T
```

1.4.2 bimacalfluxtable - Tool

Package bima
Module bimacalibrater

Tool to retrieve information from the BIMA calibrator tables

Description

A `bimacalfluxtable` allows for retrieval of information from the BIMA calibrator flux tables in the AIPS++ data repository.

Constructors

bimacalfluxtable	Construct a bimacalfluxtable tool

Functions

close	Close the table tools associated with this tool
contains	does the table contain the specified calibrator?
done	Done with this tool
fluxlist	Get the flux history for the specified calibrator
fluxtable	Get the table containing the flux histories of calibrators
nearestflux	Get the flux nearest in time to the specifed date of the specified calibrator.
positiontable	Get, as a table tool, the position table (J2000)
type	Get the type of this tool, namely 'bimacalfluxtable'

bimacalfluxtable - Constructor

Package bima
Module bimacalibrater
Tool bimacalfluxtable

Construct a bimacalfluxtable tool

Synopsis

bimacalfluxtable()

Description
Create a `bimacalfluxtable` tool.

Arguments

Returns
bimacalfluxtable tool

Example

```
# the '+' is printed by glish, not typed by you
- bcft := bimacalfluxtable();
```

---

bimacalfluxtable.contains - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

does the table contain the specified calibrator?

Synopsis
contains(calname)

Description

This function returns true if the specified calibrator is contained in the position (J2000) table.

Arguments

calname	The calibrator to determine the existance of Allowed:String Default: none

Returns
Boolean

Example

```
- bcft := bimacalfluxtable();
- bcft.contains('1733-130');
T
- bcft.contains('somedummycal');
F
```

---

bimacalfluxtable.fluxlist - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

Get the flux history for the specified calibrator

Synopsis
fluxlist(calname)

Description

This function retrieves, as a record, the flux history for the specified calibrator. The fields in the resulting record are `obsdate` (in fiducial days), `flux` (in Jy), `rms` (in Jy), and `freq` (in GHz).

Arguments

calname	Name of the calibrator for which to get the flux history Allowed:String Default: none

Returns
a record containing the history or fail

Example

```
- bcft := bimacalfluxtable();
- r := bcft.fluxlist('1733-130');
- r.flux;
[5.5999999 6.30000019 5.5 5.5999999 6 4.5999999 6 5.69999981 6.4000001
6.80000019 8.19999981 8.69999981 11.6000004 11 9.5 10 11 10 10.1999998
10.3000002 11.6000004 6.5 2.79999995 3.20000005 2.5999999 2.5
2.20000005 2.0999999 2.29999995 2.5 2.5999999 2.9000001 2.79999995
3.4000001 3.5 3.20000005 4.30000019 4.30000019]
- r.obsdate[5];
49250
```

---

bimacalfluxtable.fluxtable - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

Get the table containing the flux histories of calibrators

Synopsis
fluxtable()

Description

This function returns as a table tool the table containing calibrator flux density history.

Arguments

Returns
Table tool

Example

```
- bcft := bimacalfluxtable();
- fluxtable := bcft.fluxtable();
```

---

bimacalfluxtable.nearestflux - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

Get the flux nearest in time to the specifed date of the specified calibrator.

Synopsis
nearestflux(calname, date)

Description

This function returns, as a record, information about the specified calibrator closest to the specified date. The field names of the record returned are `obsdate` (in fiducial days), `flux` (in Jy), `rms` (in Jy), and `freq` (in GHz).

Arguments

calname	The calibrator for which to retrieve information Allowed:String Default: None
date	The date to which to get the nearest-in-time record Allowed:Quantity Default: None

Returns
Record or fail

Example

```
- bcft := bimacalfluxtable();
- bcft.nearestflux('1733-130',dq.quantity('today'));
[obsdate=52328, flux=4.30000019, rms=0.600000024, freq=86.1999969]
- bcft.nearestflux('1733-130',dq.quantity('01jan2000'));
[obsdate=51605, flux=2.0999999, rms=0.5, freq=86.1999969]
```

---

bimacalfluxtable.positiontable - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

Get, as a table tool, the position table (J2000)

Synopsis
positiontable()

Description

This function returns as a table tool the position table of BIMA calibrators.

Arguments

Returns
Table tool

Example

```
- bcft := bimacalfluxtable();
- postable := bcft.positiontable();
```

---

bimacalfluxtable.type - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

Get the type of this tool, namely 'bimacalfluxtable'

Synopsis
type()

Description

This function simply returns the string `'bimacalfluxtable'`.

Returns
'bimacalfluxtable'

Example

```
- bcft := bimacalfluxtable();
- bcft.type();
bimacalfluxtable
```

---

bimacalfluxtable.done - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

Done with this tool

Synopsis
done()

Description

Release resources used by this tool.

Returns
Boolean T

Example

```
- bcft := bimacalfluxtable();
- bcft.done();
T
```

---

bimacalfluxtable.close - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

Close the table tools associated with this tool

Synopsis
close()

Description

This function closes the table tools associated with this tool.

Arguments

Returns
Boolean T

Example

```
- bcft := bimacalfluxtable();
- bcft.close();
T
```

1.4.3 bimacalibratertest - Function

Package bima
Module bimacalibrater

Test bimacalibrater

Synopsis
bimacalibratertest(verbose, modelcaltable, modeldatatable)

Description
`bimacalibratertest` is a function to test bimacalibrater. It uses datasets in the AIPS++ data repository. The basic strategy of the tests is to fill `Miriad`

datasets (a target and calibrator source) and calibrate them. The resulting calibration table is compared to a fiducial table, and similarly the `MODEL_DATA` and `CORRECTED_DATA` columns of the calibrated target dataset are compared to a fiducial dataset. The function returns `T` if all tests pass, `F` otherwise. The filling, reaveraging, calibration, and tests take 10 minutes on an unloaded Linux system with a 1.5 GHz Pentium 4 processor and a 7200 rpm IDE hard drive (many portions of the function are I/O bound). The function will create a scratch directory in the current directory called `bimacalibratertester_work` where it will write and manipulate datasets. The datasets used are rather large, and so about 350 MB of disk space is required.

Arguments

verbose	in	Controls the amount of logger output, 0=none, higher numbers=more Allowed:Int ¿= 0 Default: 1
modelcaltable	in	Use this table as the fiducial table for comparison with the generated calibration table Allowed:Any valid table name Default: ” (uses table in the data repository)
modeldatatable	in	Use this table as the fiducial table for comparison with the generated MODEL_DATA and CORRECTED_DATA columns Allowed:Any valid table name Default: ” (uses table in the data repository)

Returns

Boolean or fail

Example

```
- include 'bimacalibrater.g';
- bimacalibratertest();
<Lots of messages regarding filling, reaveraging, calibration, and testing>
T
```

1.5 bimatester - Module

Module for testing filling, calibration, and imaging of BIMA data
include "bimatester.g"

Description

The `bimatester` module provides functions for testing various aspects of AIPS++ on BIMA data.
The facilities of the `bimatester` module are made available in Glish by issuing the following command:

```
- include 'bimatester.g'
T
```

where a hyphen precedes user input. The Glish response is indicated without the prompt.
To run all the tests, simply run the `bimatest` global function

```
- bimatest()
```

Tools

bimatester	Testing of BIMA data

Functions

bimatest	A quick way to run tests.

1.5.1 bimatester - Tool

Package	**bima**
Module	**bimatester**

Testing of BIMA data

Description
A `bimatester` tool provides all the functionality required for testing BIMA data. The tools tests data by running various functions for filling, calibrating, and imaging BIMA data. It compares the results with fiducial datasets located in the AIPS++ Data Repository. The fiducial datasets were created using a Linux installation using the GNU compiler. The filling and calibration tests can be performed on other architectures, but because the results of imaging differ significantly from Linux to Solaris, this tool will only do image tests if

run on a Linux installation.

Constructors

bimatester	Construct a bimatester tool

Functions

runtests	Run various tests
summary	Summarize the test results

bimatester - Constructor

Package	**bima**
Module	**bimatester**
Tool	**bimatester**

Construct a bimatester tool

Synopsis
bimatester(verbose)

Description
Create a `bimatester` tool.

Arguments

verbose	Controls the level of output. 0 means run silently, higher values produce more messages Allowed:Int Default: 1

Returns
bimatester tool

Example

```
# create a bimatester tool, and specify you want a lot of output
- bt := bimatester(2)
```

---

bimatester.runtests - Function

Package	**bima**
Module	**bimatester**
Tool	**bimatester**

Run various tests

Synopsis
runtests(fill, calibrate, image, image)

Description

Run various tests on BIMA data. The user can specify which tests to run. However, imaging and cleaning tests will only be run on Linux installations because the fiducial images were created using such and installation and there are significant differences between imaging on Linux and Solaris.

Arguments

fill	Fill data and run filler tests? Allowed:Boolean Default: T
calibrate	Calibrate data and run calibration tests? Allowed:Boolean Default: T
image	Image data and run imaging tests? Allowed:Boolean Default: T
image	Clean the image and run deconvolution tests? Allowed:Boolean Default: T

Returns
T if all specified tests pass, F otherwise

Example

```
- bt := bimatester(2);
# Run all available tests
```

```
+ bt.runtests();
Lots of messages
T
```

---

bimatester.summary - Function

Package	**bima**
Module	**bimatester**
Tool	**bimatester**

Summarize the test results

Synopsis
summary(verbose)

Description

This function summarizes the test results by returning a record containing the results and optionally printing a summary to the log. A value of `T` associated with a test means that it passed, `F` if it failed.

Arguments

verbose	Print results to the log? Allowed:Boolean Default: T

Returns
record of test results

Example

```
- bt := bimatester(2);
# Run all available tests
+ bt.runtests();
Lots of messages
T
- results := bt.summary()
```

```
NORMAL: BIMATESTER.FILLER: T
NORMAL: BIMATESTER.CALIBRATION: T
NORMAL: BIMATESTER.IMAGING: T
NORMAL: BIMATESTER.CLEANING: T
- print results;
[FILLER=T, CALIBRATION=T, IMAGING=T, CLEANING=T]
```

1.5.2 bimatest - Function

Package **bima**
Module **bimatester**

A quick way to run tests.

Synopsis
bimatest(verbose, summarize, fill, calibrate, image, image)

Description
This global function runs user-specified tests on BIMA data by creating a `bimatester` tool, running the tests, and optionally prining a summary.

Arguments

verbose	Controls the level of output. 0 means run silently, higher values produce more messages Allowed:Int Default: 1
summarize	Summarize the results? Allowed:Boolean Default: T
fill	Fill data and run filler tests? Allowed:Boolean Default: T
calibrate	Calibrate data and run calibration tests? Allowed:Boolean Default: T
image	Image data and run imaging tests? Allowed:Boolean Default: T
image	Clean the image and run deconvolution tests? Allowed:Boolean Default: T

Returns
T if all tests passed, F otherwise

Example

```
- include 'bimatester.g';
- bimatest();
<Lots of messages>
T
```

Index