1.3 ms - Module

Module for measurement set operations
include "ms.g"

Description
An AIPS++ measurement set is a AIPS++ table which obeys specific conventions. These conventions are defined in note 229. Like all AIPS++ tables the measurement set will always appear as a directory which contains a number of files and directories.
Measurement set tables come in two slightly different versions, single dish and interferometric. Single dish measurement sets store the observed data as real numbers in the `FLOAT_DATA` column of the measurement set, whereas interferometric ones use complex numbers in the `DATA` column.
A measurement set table can contain data from a variety of different observations with different spectral or polarimetric configurations, different pointings and different instruments. To do this it needs to handle data with differing shapes. The data shape referred to here is two-dimensional with the length of the axes being the number of correlations and the number of channels in the data. A typical shape might be `[4, 1]`, which could correspond to a continuum observation where the `[RR, LL, RL, LR]` polarizations where correlated. In the same measurement set there may be data with a shape of `[1, 32]`, which corresponds to a spectral line observation, with 32 channels, where only the `[XX]` polarizations are correlated.

Tools

ms	Operations on measurement sets
ms2fromms1	MeasurementSet Version 1 to Version 2 converter
msplot	tool for MeasurementSet plotting, display, listing and editing

Functions

is_ms	Is the argument a ms tool?
msfiles	Find the list of MS files in the given directory
msplotapplyflags	Apply flag columns
mstest	Test for the proper functioning of the ms `tool`

1.3.1 ms - Tool

Package general
Module ms

Operations on measurement sets
include "ms.g"

Constructors

fitstoms	Create an ms tool from a UVFITS file
ms	Create an ms tool attached to a measurement set
sdfitstoms	Create an ms tool from a SDFITS file

Functions

clearbuffer	Clear the internal buffer.
clipbuffer	Clip the internal buffer with specified limits.
close	Detach the ms `tool` from the measurement set table
command	Construct a ms `tool` using a table query command
concatenate	Concatenate two measurement sets
continuumsub	Continuum fitting and subtraction in uv plane
createflaghistory	Create and initialize the FLAG_HISTORY column.
diffbuffer	Differentiate or difference the internal buffer.
done	Delete the ms `tool`
fillbuffer	Fill the internal buffer with data and flags.
flaglevel	return the current flagging level
getbuffer	Return the internal buffer as a Record for access from Glish.
getdata	Read values from the measurement set.
iswritable	Returns True is the underlying Table is writable
iterend	End the iteration and reset the selected table
iterinit	Initialize for iteration over a MS
iternext	Advance the iterator to the next lot of data
iterorigin	Set the iterator to the start of the data.
lister	List the measurement set's data in a concise form
listhistory	List history of the measurement set
name	Name of the measurement set table the tool is attached to.
nrow	Returns the number of rows in the measurement set
open	Attach the ms `tool` to a measurement set table
ptsrc	Reports global average of visibilities per field
putdata	Write new values into the measurement set
range	Get the range of values in the measurement set
restoreflags	Restore the specified flagging level from the FLAG_HISTORY column.
saveflags	Save the current flagging state to the FLAG_HISTORY column.
select	Select a subset of the measurement set.
selectchannel	Select and average frequency channels
selectinit	Initialize the selection of a MS
selectpolarization	Select and convert of polarizations
selecttaql	Select a subset of the measurement set.
setbufferflags	Set the flags in the buffer
split	make a new ms from a subset of actual ms

summary	Summarize the measurement set
tofits	Convert a measurement set to a UVFITS file
tosdfits	Convert a single dish AIPS++ measurement set to a SDFITS file
uvlsf	Continuum fitting and subtraction in uv plane
writebufferflags	Write the flags in the internal buffer back to the table.
writehistory	Add a row of arbitrary information to the measurement set history table

Description

The ms `tool` is provides functions to manipulate the contents of measurement set tables.
It is easy to get an ms `tool` and a measurement set table confused. The table is the disk files that contain your data. The tool is a process on your computer that provides functions that allow you access the data in the table. One tool can access only one table at any time. If you want to access two measurement sets simultaneously you need to create two ms `tools`.
The ms `tool` is frequently used as a server of data for higher-level `tools` such as visplot.
The ms `tool` functions can be categorised as shown below.

Creating a ms `tool`
A ms `tool` is usually associated with a measurement set table. The simplest and most common way to make a ms tool is to use the ms constructor which requires you to specify the name of the measurement set table.
There are two other constructors which convert FITS files to a measurement set table prior to creating the ms tool (which is attached to the newly created measurement set — not the FITS file). These are the fitstoms constructor, which reads a UVFITS file, and the sdfitstoms constructor, which reads a single dish FITS file. The conversion step may take some time if the FITS file is large. However it only needs to be done once. The measurement set table is not deleted when you delete the ms tool, using the done function, or exit AIPS++. And once the measurement set table is created it is much faster to create an ms tool using the ms constructor.

Getting summary information
The summary function will display, in the logger, an overview of the measurement set. This will include listings of the fields, spectral windows & polarization setups used in the measurement set.
The range function will provide more quantitative information on the minimum, maximum or used values of specified parameters. When using this function you may need to do an initial selection, as described below, depending on whether the parameters you ask for change their shape. A list of parameters accepted by the range function is given in table 1.355 and this table also indicates when an initial selection is necessary.
The lister function provides a concise listing of the data in the measurement set.

The listhistory function lists the contents of the measurement set history table. The history table contains a record of changes made to the measurement set by autoflag, calibrater, imager and other tools.

Selecting data

As described in the ms module documentation a measurement set can contain data with a variety of different shapes. Some of the functions in this tool require the data to be in a fixed shape. Before you can use these functions you need to select a subset of the data in the measurement set where all the data has a fixed shape. There are two functions which can be used to do this. These are the selectinit and command functions.

The select function can be used to further refine which subset of the data will be used by the data access functions. This function allows you to select specific rows in a measurement set using a wide range of criteria.

The select function can only select whole rows in a measurement set. To select specific channels within a row you use the selectchannel function. Similarly to select specific polarizations you should use the selectpolarization function.

Reading and writing data

The getdata function is used to read data from the measurement set into *Glish* variables. You can select which columns of the measurement set main table you are interested in and only the subset of data specified using the selection functions described above will be retreived. Any frequency averaging (see the selectfrequency function) and polarization conversion (see the selectpolarization function) will be done when you retreive the data. The full power of *Glish* and other AIPS++ tools, like pgplotter, can then be used for adhoc inspection and calculations involving the data.

If the measurement set was opened for writing then the putdata function can be used to write the data back into the measurement set. When writing data back into the measurement set you cannot change the data shape or the coordinates of the data, only the numerical values. This means that you cannot write data that has been averaged in frequency or converted to different polarizations.

When using the getdata function with a large measurement set you need to be careful to not request too much data. The measurement set is stored on disk but *Glish* variables are stored in memory. To allow you to access large amounts of data in an ordered way the ms tool provides functions that allow you iterate through the data is convienient way.

If you need to step through the data in an orderly fashion, you can use the iteration functions. These allow you to set up an iteration order (iterinit), obtain the first iteration (iterorigin), go to the next iteration (iternext) and end the iteration prematurely (iterend). The iterorigin and iternext function set the currently selected table (as used by getdata and others) to the current iteration. At the end of the iteration, the original selection is restored.

You can iterate through a measurement set you have previously selected using select, but if you use select while iterating, you cannot get back the unselected iteration (without reiterating through the table until the current point).

The writehistory function allows messages to be appended to the measurement set history table should the you wish to do so. The listhistory function lists your messages and those created by autoflag, calibrater, imager and other tools.

Conversions to FITS
Just as the fitstoms constructor will convert a UVFITS file to a measurement set the tofits function will convert a measurement set to a UVFITS file. Similarly a single dish measurement set ie., one with a `FLOAT_DATA` column rather than a `DATA` column, can be converted to a single dish FITS file using the tosdfits function.
You cannot read a UVFITS file into a measurement set and write it out as a single dish FITS file or vice-versa.

Concatenation
The concatenate function can be used to append the data from one measurement set to the end of another. As all the data is copied this function may take some time if the measurement set to be copied is large. The measurement set needs to be opened for writing for this to work.

Splitting The split function allow you to make a new ms from a subset of the actual ms

Flagging data
The flag and buffer functions all belong together. The idea is to fill a buffer of data (fillbuffer), optionally retrieve it as a record (getbuffer) and display it in some data display tool, do operations like differencing (diffbuffer), clipping (clipbuffer) and manual edits of the displayed data (setbufferflags), and then write the flags back to the MS (writebufferflags), into the appropriate flag level, so you can choose to apply or undo them. The data is untouched by these functions.

Table 1.355: Items recognized by the range, select, getdata and putdata functions. Items marked with a † are only available in interferometric measurement sets. Items marked with a ‡ are only available in interferometric measurement sets that have been processed with calibrator or imager. Items marked with a $*$ do not require all the data in the selected measurement set is the same shape.

	range	select	getdata	putdata	comment
amplitude†	+	-	+	-	amplitude of observed data
corrected_amplitude‡	+	-	+	-	amplitude of corrected data
model_amplitude‡	+	-	+	-	amplitude of model data
ratio_amplitude‡	-	-	+	-	amplitude of corrected/model
residual_amplitude‡	-	-	+	-	amplitude of residual data
obs_residual_amplitude‡	-	-	+	-	amplitude of obs residual data
antenna1	$*$	+	+	-	1st antenna id

Continued from previous page					
	range	select	getdata	putdata	comment
antenna2	*	+	+	-	2nd antenna id
antennas	*	-	-	-	list of antenna names
array_id	*	+	-	-	
axis_info	-	-	+	-	description of data axes
chan_freq	+	-	-	-	channel frequencies
corr_names	+	-	-	-	list of polarization strings
corr_types	+	-	-	-	list of polarization enum values
data†	-	-	+	+	complex observed data
corrected_data‡	-	-	+	+	complex corrected data
model_data‡	-	-	+	+	complex model data
ratio_data‡	-	-	+	-	complex corrected/model
residual_data‡	-	-	+	+	complex residual data
obs_residual_data‡	-	-	+	+	complex observed residual data
feed1	*	+	+	-	1st feed in correlation
feed2	*	+	+	-	2nd feed in correlation
field_id	*	+	+	-	field number
fields	*	-	-	-	list of field names
flag	-	-	+	+	data flags
flag_row	-	-	+	+	MS row flags
flag_sum	-	-	+	-	flag summary
ha	-	-	+	-	add hour angle to axis_info
ifr_number	*	+	+	-	1000*antenna1+antenna2
imaginary†	+	-	+	-	imag part of observed data
corrected_imaginary‡	+	-	+	-	imag part of corrected data
model_imaginary‡	+	-	+	-	imag part of model data
ratio_imaginary‡	-	-	+	-	imag part of corrected/model
residual_imaginary‡	-	-	+	-	imag part of residual data
obs_residual_imaginary‡	-	-	+	-	imag part of obs residual data
imaging_weight‡	+	-	+	+	weights used for imaging
last	-	-	+	-	add LAST to axis_info
num_corr	+	-	-	-	number of polarizations
num_chan	+	-	-	-	number of freq channels
phase†	+	-	+	-	phase of observed data
corrected_phase‡	+	-	+	-	phase of corrected data
model_phase‡	+	-	+	-	phase of model data
ratio_phase‡	-	-	+	-	phase of corrected/model
residual_phase‡	-	-	+	-	phase of residual data
obs_residual_phase‡	-	-	+	-	phase of observed residual data
phase_dir	+	-	-	-	list of phase centers & epoch
real†	+	-	+	-	real part of observed data
corrected_real‡	+	-	+	-	real part of corrected data
model_real‡	+	-	+	-	real part of model data
ratio_real‡	-	-	+	-	real part of corrected/model
residual_real‡	-	-	+	-	real part of residual data

Continued from previous page					
	range	select	getdata	putdata	comment
obs_residual_real‡	-	-	+	-	real part of observed res. data
ref_frequency	+	-	-	-	reference frequency
rows	*	+	-	-	row numbers in *original* table
scan_number	*	+	+	-	
sigma	*	-	+	+	sigma of the data
data_desc_id	*	+	+	-	
time	*	+	+	-	MJD time range in seconds
times	*	+	-	-	list of MJD timeslots
ut	-	-	+	-	add UT to axis_info
uvw	-	-	+	-	uvw vector
u	*	+	+	-	u coordinate
v	*	+	+	-	v coordinate
w	*	+	+	-	w coordinate
uvdist	*	+	+	-	uv distance
weight	*	-	+	+	weight of the data

Example

```
include 'ms.g'
include 'statistics.g'
myms:=ms("3C273XC1.MS",readonly=F);
myms.selectinit(datadescid=2)
myms.select([antenna1=[1,3,5],uvdist=[1200,1900]]);
rec:=myms.getdata("weight corrected_data");
rec.weight*:=1.2;
rec.corrected_data-:=mean(rec.corrected_data);
myms.putdata(rec);
myms.close();
```

We open the MS for writing, select an array and spectral window and then select a few antennas and a uv range. We then get out the weight values and the corrected_data. We change these values in Glish and then write them back to the measurement set. Finally, we close the ms, causing the values to be written back to disk. Note: you cannot use the name ms as a Glish variable (e.g., ms:=ms("MyMS")), because that would overwrite the ms definition itself.

ms - Constructor

Package	**general**
Module	**ms**
Tool	**ms**

Create an ms tool attached to a measurement set

Synopsis
ms(filename, readonly, lock, host, forcenewserver)

Arguments

filename	in	measurement set file-name Allowed:String Default: no default
readonly	in	open for read access only Allowed:Boolean Default: T
lock	in	lock the table for exclusive use Allowed:Boolean Default: F
host	in	host to start ms tool on Allowed:String Default: ”
forcenewserver	in	start a new server tool Allowed:Boolean Default: F

Returns
ms `tool` or fail

Description
This is the most commonly used constructor. It creates an ms tool which is attached to the specified measurement set table.
By default the table is opened read only to prevent you from accidently making changes to the measurement set. Set readonly to F you you do intend to make changes.
Setting the lock argument to T will permanently lock the table preventing other processes from writing to the measurement set. Unless you expect this to happen, and want to prevent it, you should leave the lock argument at the default value which implies auto-locking.
The host argument specifies which machine the precompiled ms process should be started on. The default value starts it on the same machine as the one that *Glish* is running on.
In order to run the ms tool on a remote machine you need to satisfy all the following conditions.

- It must be possible to start AIPS++ on the remote machine

- You must be able to log onto the remote machine without having to type a password
- The AIPSPATH environment variable must be defined on the remote machine. You may want to set this up in the relevant "dot" file eg., adding a line like `source /usr/local/aips++/aipsinit.csh` in your .cshrc file (for csh).

One quick way to check if all three conditions are met is to type, on your local machine `rsh host 'echo $AIPSPATH'` where host is replaced by the name of the remote machine. If the value of the AIPSPATH variable that is printed does not contain something like `aips-root architecture site host` and that all the values are correct for the remote machine you can be certain that starting the ms tool, or any AIPS++ server, on the remote host will not work
Each ms tool can only run one function at a time. To solve this you start two servers. The forcenewserver argument allows you to do this by overriding the default behaviour of having each ms tool share the same server.
This function returns an ms tool or fail if something went wrong, like an error in the measurement set name.

Example
See the example for the nrow function.

---

fitstoms - Constructor

Package	**general**
Module	**ms**
Tool	**ms**

Create an ms tool from a UVFITS file

Synopsis
fitstoms(msfile, fitsfile, readonly, lock, obstype, host, forcenewserver)

Arguments

msfile	in	Filename for the newly created measurement set Allowed:String Default: no default
fitsfile	in	UVFITS file to read Allowed:String Default: no default
readonly	in	open for read access only Allowed:Boolean Default: T
lock	in	lock the table for exclusive use Allowed:Boolean Default: F
obstype	in	specify the observation type: 0=standard, 1=fastmosaic, requiring small tiles in the measurementset Allowed:Int Default: 0
host	in	host to start ms tool on Allowed:String Default: ”
forcenewserver	in	start a new server tool Allowed:Boolean Default: F

Returns
ms tool or fail

Description
This constructor will convert a UVFITS file to a measurement set table and then create a ms tool that is attached to the measurement set table. The newly created measurement set table will continue to exist after the tool has been shutdown.
The last four arguments of this constructor are identical in behaviour to the ms constructor. See the documentation for that function for more details on their meaning.
Note that the variety of fits files that fitstoms is able to interpret correctly is limited mostly to files similar to those produced by classic AIPS. In particular, it understands only binary table extensions for the antenna (AN), frequency (FQ) and source (SU) information and ignores other extensions.
This function returns an ms tool or fail if something went wrong, like an error in a file name.

Example

```
include 'ms.g'
myms := fitstoms("3C273XC1.ms", "3C273XC1.fits")
```

---

sdfitstoms - Constructor

Package	**general**
Module	**ms**
Tool	**ms**

Create an ms tool from a SDFITS file

Synopsis
sdfitstoms(msfile, fitsfile, readonly, lock, host, forcenewserver)

Arguments

msfile	in	Filename for the newly created measurement set Allowed:String Default: no default
fitsfile	in	SDFITS file to read Allowed:String Default: no default
readonly	in	open for read access only Allowed:Boolean Default: T
lock	in	lock the table for exclusive use Allowed:Boolean Default: F
host	in	host to start ms tool on Allowed:String Default: ”
forcenewserver	in	start a new server tool Allowed:Boolean Default: F

Returns
MS tool

Description
This constructor will convert a SDFITS file to a measurement set table and

then create a ms tool that is attached to the measurement set table. The newly created measurement set table will continue to exist after the tool has been shutdown.
The SDFITS binary table must be the first extension in the indicated FITS file. A brief description of SDFITS may be found in the **dish** users manual. The last four arguments of this constructor are identical in behaviour to the ms constructor. See the documentation for that function for more details on their meaning.
This function returns an ms tool or fail if something went wrong, like an error in a file name.

Example

```
include 'ms.g'
myms := sdfitstoms("dishdemo1.ms", "dishdemo1.fits")
```

---

ms.nrow - Function

Package	**general**
Module	**ms**
Tool	**ms**

Returns the number of rows in the measurement set

Synopsis
nrow(selected)

Arguments

selected	in	return number of selected rows Allowed:Boolean Default: F

Description
This function returns the number of rows in the measurement set. If the optional argument selected is set to T, it returns the number of currently selected rows, otherwise it returns the the number of rows in the original measurement.

Returns
Integer

Example

```
include 'ms.g'
myms := ms('3C273XC1.ms')
print "Number of rows in ms =", myms.nrow()
```

---

ms.iswritable - Function

Package	**general**
Module	**ms**
Tool	**ms**

Returns True is the underlying Table is writable

Synopsis
iswritable()

Description
This function returns T(rue) if the underlying MeasurementSet Table was opened for writing/update.

Returns
Boolean

Example

```
include 'ms.g'
myms := ms('3C273XC1.ms',readonly=F)
if (myms.iswritable()) {
  print "MeasurementSet is writable";
} else {
  print "MeasurementSet is readonly";
}
```

---

ms.open - Function

Package	**general**
Module	**ms**
Tool	**ms**

Attach the ms `tool` to a measurement set table

Synopsis
open(thems, readonly, lock)

Arguments

thems	in	Name of the measurement set table to open Allowed:String Default: no default
readonly	in	prevent changes to the measurement set Allowed:Boolean Default: T
lock	in	lock the table for exclusive use by this tool Allowed:Boolean Default: F

Returns
nothing

Description
Use this function when you have detached (using the close function) an existing ms `tool` from a measurement set table and wish to reattach to another measurement set table. This is quicker than destroying the ms `tool` with the done function and starting up a new one as the server does not have to be restarted.

Example

```
include 'ms.g'
myms := ms('3C273XC1.ms')
myms.close();
myms.open("anotherms", readonly=F, lock=F)
```

---

ms.close - Function

Package	**general**
Module	**ms**
Tool	**ms**

Detach the ms `tool` from the measurement set table

Synopsis
close()

Description
This function detaches the ms `tool` from the associated measurement set table after flushing all the cached changes. After calling this function the ms `tool` is not associated with any measurement set and using any function other than open will result in an error message being sent to the logger.
This function can be useful to avoid synchronization problems which can occur when different processes have the same ms open.

Returns
nothing

Example
See the example for the open function.

---

ms.name - Function

Package	**general**
Module	**ms**
Tool	**ms**

Name of the measurement set table the tool is attached to.

Synopsis
name()

Description
This function returns the name of the measurement set table that is being

manipulated. If the ms `tool` is not attached to any measurement set then this function will return the string “none”.

Returns
String

Example

```
include 'ms.g'
myms := ms('3C273XC1.ms')
print "Processing file", myms.name()
```

---

ms.command - Function

Package	**general**
Module	**ms**
Tool	**ms**

Construct a ms `tool` using a table query command

Synopsis
command(msfile, command, readonly)

Arguments

msfile	in	name of selected measurement set table Allowed:String Default: no default
command	in	Query string Allowed:Any valid table command Default: no default
readonly	in	prevent changes to the measurement set Allowed:Bool Default: T

Returns
ms tool or fail

Description
Make a new measurement set table and a new ms `tool`, from a subset of the data in the measurement set the current ms `tool` is attached to. The subset is selected using the qualification part of the table command as described in note 199. This function will return a new ms `tool`.
To use this function successfully it helps to have an understanding of the column names used in a measurement set table. These are described in note 229. Any identifiers that are used to refer to rows in a subtable will be zero based ie., the first row is row zero and not row one. This is different to the convention used throught *Glish* and in other parts of this tool. The example below illustrates this.
The new measurement set must be given a name but it is not written to disk until the close or done functions are performed by the associated tool.
The resulting measurement set is always a reference to the original measurement set, therefore you should not delete the original measurement set from disk unless you also intend to deleted the selected one. However the converse is not true, you may delete the selected measurement set and keep the original one.

Example

```
include 'ms.g'
myms := ms('3C273XC1.ms')
subms := myms.command('3C273XC1-dd1.ms', 'DATA_DESC_ID == 0')
print 'The original ms has', myms.nrow(), 'rows.'
print 'The selected one has', subms.nrow(), 'rows.'
subms.close()
```

In this example a new ms `tool`, called subms, is created that accesses a newly created measurement set called `3C273XC1-dd1.ms`. The new tool provides access only to the data where the value in the `DATA_DESC_ID` column is zero. This is equivalent to using the selectinit function with an argument of one. This difference in indexing value, between the command and the selectinit functions is illustrative of the difference between zero and one based indexing. If the measurement set you are using contains data with different spectral windows or polarization setups, then the selected measurement set will have less rows than the original measurement set.

Example

```
include 'ms.g'
myms := ms('3C273XC1.ms')
```

```
subms := myms.command('3C273XC1-a12.ms',
                      'ANTENNA1 == 0 && ANTENNA2 == 1')
print 'The original ms has', myms.nrow(), 'rows.'
print 'The selected one has', subms.nrow(), 'rows.'
subms.close()
```

This is identical to the previous example except that a more complex command has been used. In this case all the data from one baseline are selected. The baseline chosen corresponds to data collected from the antennas defined in the first and second rows of the antenna sub-table of the measurement set.

Example

```
include 'ms.g'
myms := ms('3C273XC1.ms')
subms := myms.command('3C273XC1-a12.ms',
                      'ANTENNA1 == ANTENNA2', readonly=F)
subms.
subms.close()
```

This example selects all the auto-correlations in the measurement set. The resultant ms tool will allow you to modify the data. Because the selected measurement set references the data in the original measurement set modifying the data in the selected measurement set will also change the auto-correlations in original measurement set.

---

ms.tofits - Function

Package	**general**
Module	**ms**
Tool	**ms**

Convert a measurement set to a UVFITS file

Synopsis
tofits(fitsfile, column, fieldid, spwid, start, nchan, width, writesyscal, multisource, combinespw, writestation)

Arguments

fitsfile	in	Name of the new UVFITS file Allowed:String Default: no default
column	in	Data column to write, see above for options Allowed:String Default: corrected
fieldid	in	selection of fields to write out Allowed:Vector of ints Default: []
spwid	in	selection of spectral windows to write out Allowed:Vector of ints Default: []
start	in	First channel at output Allowed:Int Default: -1
nchan	in	number of channel to output Allowed:Int Default: -1
width	in	number of input channels to average for output Allowed:Int Default: -1
writesyscal	in	Write GC and TY tables Allowed:Boolean Default: F
multisource	in	Write in multisource format Allowed:Boolean Default: F
combinespw	in	Handle spectral window as IF Allowed:Boolean Default: F
writestation	in	Write station name instead of antenna name Allowed:Boolean Default: F

Description

This functions writes a UVFITS file that contains the data in the measurement set associated with this tool. The FITS file is always written in floating point format and the data is always stored in the primary array of the FITS file.
If the measurement set has been processed by many of the imaging tools, including imager and calibrator, it may contain additional data columns. You need to select which of these data columns will be written to the FITS file. The possible options are:

observed This is the raw data as collected by the telescope. All interferometric measurement sets must contain this column. A synonym for 'observed' is 'data'.

corrected This is the calibrated data. A synonym for 'corrected' is 'corrected_data'.

model This is the visibilites that would be measured using the current model of the sky. A synonym for 'model' is 'model_data'.

The parsing of these strings is case insensitive. If any other option is specified then the observed data will be written.
By default a single-source UVFITS file is written, but if the measurement set contains more than one field or if you set the multisource argument to T a multi-source UVFITS file will be written. Because of limitations in the UVFITS format you have to ensure that the data shape is fixed for all the data you intend to write to one FITS file. See the general description of this tool for how you can select data to meet this condition.
The combinespw argument is used to control whether data from different spectral windows will be written as different entries in the FITS FQ (frequency) table or combined as different IF's within one entry in the FQ table. You should set this to T only if you know that the data from different spectral windows was observed simultaneously, and the data in the measurement set can be equally divided between all the spectral windows. Use of this switch is recommended for data to be processed in classic AIPS (if possible, e.g., standard dual IF observations).
The FITS GC (gain curve) and TY (system temperature) tables can be optionally written by setting the writesyscal argument to T. This is a rather WSRT-specific operation at the moment and may not work correctly for measurement sets from other telescopes.
The three parameters 'start', 'nchan', 'width' are for channel averaging while outputing the data to the fits file. The default values of -1 will copy the channels of the input as is. 'start' is input channel number that will be the first channel of the output. 'nchan' is the number of channel wanted in the output data. 'width' is the number of channels of the input data to make 1 channel of the output data. Hence (nchan*width+start-1) should not be larger than the number of channels in the original data.

Example

```
include 'ms.g'
myms := ms('3C273XC1.ms')
myms.tofits('3C273XC1.fits', column='DATA');
myms.done()
```

This example writes the observed data of a measurement set to a UVFITS file.

```
include 'ms.g'
myms := ms('big.ms')
myms.tofits('part.fits', column='CORRECTED', fieldid=[1,2], spwid=[3]);
myms.done()
```

This example writes part (the first 2 fields and for the third spectral window) of the corrected data to the fits file.

---

ms.summary - Function

Package	**general**
Module	**ms**
Tool	**ms**

Summarize the measurement set

Synopsis
summary(header, verbose)

Arguments

header	out	Header information Allowed:Record Default: empty record
verbose	in	Produce verbose logging output Allowed:Boolean Default: F

Description
This function will print a summary of the measurement set to the system logger. The verbose argument provides some control on how much information is displayed.
This function can also return, in the header argument, a record containing the following fields.

nrow Number of rows in the measurement set

name Name of the measurement set

Example

```
include 'ms.g'
myms := ms('3C273XC1.ms')
myms.summary(verbose=T)
```

This example will send a verbose summary of the measurement set to the logger.

---

ms.listhistory - Function

Package	**general**
Module	**ms**
Tool	**ms**

List history of the measurement set

Synopsis
listhistory()

Description
This function lists the contents of the measurement set history table.

Example

```
# Make test MS
include 'imager.g'
imagermaketestms('3C273XC1.ms')
# List history of test MS
include 'ms.g'
myms := ms('3C273XC1.ms')
# Turn off pager
system.output.pager.limit:=-1
myms.listhistory()
```

The history table contents are listed.

---

ms.writehistory - Function

Package	**general**
Module	**ms**
Tool	**ms**

Add a row of arbitrary information to the measurement set history table

Synopsis
writehistory(message, parms, origin, msname, app)

Arguments

message	in	Message to be recorded in message field Allowed: Any valid string Default: no default
parms	in	String to be written to input parameter field Allowed: Any valid string Default: ”
origin	in	String to be written to origin field Allowed: Any valid string Default: 'ms::writehistory()'
msname	in	name of selected measurement set Allowed: Any valid measurement set name; defaults to currently open measurement set Default: ”
app	in	String to be written to application field Allowed: Any valid string Default: 'ms'

Description
This function adds a row to the history table of the specified measurement set containing any message that the user wishes to record. By default the history entry is written to the history table of the measurement set that is currently open, the message origin is recorded as 'ms::writehistory()', the originating application is 'ms' and the input parameters field is empty.

Example

```
# Make test MS
include 'imager.g'
imagermaketestms('3C273XC1.ms')
# Write history entry to the test MS
include 'ms.g'
myms := ms('3C273XC1.ms')
myms.writehistory('an arbitrary history message')
# Turn off pager and list the history of MS
```

```
system.output.pager.limit:=-1
mymms.listhistory()
```

A row is appended to the measurement set history table.

---

ms.range - Function

Package	**general**
Module	**ms**
Tool	**ms**

Get the range of values in the measurement set

Synopsis
range(items, useflags, blocksize)

Arguments

items	in	Item names Allowed:Vector of strings Default: no default
useflags	in	Use the data flags Allowed:Bool Default: T
blocksize	in	Set the blocksize in MB Allowed:Int Default: 10

Returns
record with ranges for the requested items

Description
This function will return the range of values in the currently selected measurement set for the items specified. Possible items include most scalar columns, interferometer number (1000*antenna1+antenna2), uvdist(ance), u, v, w, amplitude, phase, real and imaginary components of the data (and corrected and model versions of these - if these columns are present). See the table at the top of the document to find the exact list.
You specify which items you are interested in using a string vector where each element is a case insensitive item name. This function will then return a

record that has fields corresponding to each of the specified items. Each field will contain the range of the specified item. For many items the range will be the minimum and maximum values but for some it will be a list of unique values. Unrecognized items are ignored.
By default the FLAG column is used to exclude flagged data from before any ranges are determined, but you can set useflags=F to include flagged data in the range (but if you average in frequency, flagging will still be applied).
You can influence the memory use and the reading speed using the blocksize argument - it specifies how big a block of data to read at once (in MB). For large datasets on machines with lots of memory you may speed things up by setting this higher than the default (10 MB).
For some items, as indicated with an § in table 1.355 (in the general description of this tool), you need to call selectinit to select a portion of the data with a unique shape prior to calling this function.
Items prefixed with either; corrected, model, residual or obs_residual and the imaging_weight item are not available unless your measurement set has been processed either with the imager or calibrator tools.

Example

```
include 'ms.g'
myms:=ms("3C273XC1.ms");
myms.selectinit(datadsescid=1);
a.range("time uvdist amplitude antenna1")
[time=[4.12163e+09 4.12164e+09] ,
 uvdist=[46.2691 3727.97] ,
 amplitude=[0.0260339 33.8518] ,
 antenna1=[1 2 3 4 ... 27] ]
```

In this example the minimum and maximum observation times, uvdistances, data amplitudes are returned as well as a list of all the antennas in the antenna1 column. Note that the double quoted string produces a string vector as the input argument.
For this dataset the selectinit function did not need to be called as all the data is of one shape.

---

ms.lister - Function

Package	**general**
Module	**ms**
Tool	**ms**

List the measurement set's data in a concise form

Synopsis
lister(starttime, stoptime)

Arguments

starttime	in	Timestamp to begin listing Allowed:String Default: no default
stoptime	in	Timestamp to end listing Allowed:String Default: no default

Description
This function provides rudimentary capabilities for listing the contents of an interferometric measurement set. It will list, in the logger, the timestamps; antenna numbers; uvdist (wavelengths at the reference frequency); field, spectral window, and channel numbers (as necessary); and the amplitude (Jy), phase (degrees), and weight for each polarization that occur within the timerange indicated in the arguments. In the current release, selection by other data parameters is unavailable (lister is not yet bound to the select, selectinit, selectchannel, and selectpolarization functions in the ms tool), but this will be added soon, along with control of the precision of the listed values, and additional listing format choices, including matrix listings and observational sequence listings. Handling of single dish measurement sets will also be provided.

Example

```
include 'ms.g'
myms := ms('3C273XC1.ms')
myms.lister(starttime='27-Jun-1989/01:03:25',stoptime='27-Jun-1989/01:03:40');
```

This example will yeild the following listing in the logger (after a brief summary):

```
Date/Time:                 RR:              RL:              LR:              LL:
1989/06/27/  Intrf UVDist     Amp   Phs  Wt    Amp   Phs  Wt    Amp   Phs  Wt    Amp   Phs  Wt
------------|-----|------|-----------------|----------------|----------------|----------------
```

```
  01:03:30.0  1- 2  55956: 29.169   1.6   1  1.104-115.8   1  1.860 143.7   1 28.943   1.8   1
  01:03:30.0  1- 3  28570: 31.134  -2.2   1  1.794 -90.7   1  1.505 154.5   1 31.282  -2.7   1
  01:03:30.0  1- 4  17667: 28.938  -4.3   1  2.139-111.2   1  0.433 160.7   1 28.879  -4.3   1
  01:03:30.0  1- 5  31047: 30.944   2.9   1  1.472-109.0   1  1.545 138.9   1 30.980   2.5   1
  01:03:30.0  1- 6  38520: 31.195  -1.9   1  1.550 -97.3   1  1.485 145.1   1 31.200  -1.9   1
  01:03:30.0  1- 7  69127: 30.089   1.6   1  3.415 -92.9   1  3.120 118.9   1 29.968   1.5   1
.
.
.
```

For measurement sets with multiple fields, spectral windows, or channels, additional columns will be added to the listing.

---

ms.selectinit - Function

Package	**general**
Module	**ms**
Tool	**ms**

Initialize the selection of a MS

Synopsis
selectinit(datadescid, reset)

Arguments

datadescid	in	Data description id Allowed:integer Default: 0
reset	in	Reset to unselected state Allowed:Boolean Default: F

Returns
Boolean

Description
A measurement set can contain data with a variety of different shapes (as described in the overall description to this tool). To allow functions to return data in fixed shape arrays you need to select, using this function, rows that

contain the same data shape. You do not need to use this function if all the data in your measurement set has only one shape.
The `DATA_DESC_ID` column in the measurement set contains a value that maps to a particular row in the `POLARIZATION` and `SPECTRAL_WINDOW` subtables. Hence all rows with the same value in the `DATA_DESC_ID` column must have the same data shape. To select all the data where the `DATA_DESC_ID` value is N you call this function with the datadescid argument set to $N + 1$. The examples below make this a bit clearer.
It is possible to have a measurement set with differing values in the `DATA_DESC_ID` column but where all the data is a fixed shape. For example this will occur if the reference frequency changes but the number of spectral channels is fixed. In cases like this all the data can be selected, using this function with an argument of zero. If the data shape does change and you call this function with an datadescid set to zero it is equivalent to calling this function with datadescid set to one. When this happens the return value will be F. In all other cases it will be T.
To return to the completely unselected measurement set, set the reset argument to True. This will allow you to acces the full range of rows in the measurement set, rather than just the selected measurement set.
The datadescid must always be a non-negative integer.

Example

```
include 'ms.g'
myms := ms("3C273XC1.ms");
myms.selectinit(datadescid=1);
print myms.range("uvdist");
myms.selectinit(reset=T);
print myms.range("uvdist");
```

In this example displays the range of uv distances for the data in the specified measurement set. The first print statement will only use data where the `DATA_DESC_ID` column is 0. This will correspond to a specific spectral window and polarization setup. The second print statement will print the range of uv distances for all the data in measurement set.

---

ms.select - Function

Package	**general**
Module	**ms**
Tool	**ms**

Select a subset of the measurement set.

Synopsis
select(items)

Arguments

items	in	record with fields contain ranges and enumerations Allowed:Record Default: no default

Returns
Boolean

Description
This function will select a subset of the current measurement set based on the range of values for each field in the input record. The range function will return a record that can be altered and used as the argument for this function. A successful selection returns T. Allowable fields are tabulated in table 1.355 (in the general description of this tool). Unrecognized fields are ignored.
You need to call selectinit before calling this function. If you haven't then selectinit will be called for you with default arguments.
Repeated use of this function, with different arguments, will further refine the selection, resulting in a successively smaller selected measurement set. If the selected measurement set does not contain any rows then this function will return F and send a warning message in the logger. Otherwise this function will return T. To undo all the selections you need to use the selectinit function (with reset=T).

Example

```
include 'ms.g'
myms := ms("3C273XC1.ms");
myms.selectinit(datadescid=1)
myms.select([antenna1=[1,3,5],uvdist=[1200,1900]]);
myms.select([time=[4121629420, 4121638290]]);
include 'quanta.g';
start := dq.getvalue(dq.convert(dq.quantity('1989/06/27/01:03:40'),'s'));
stop := dq.getvalue(dq.convert(dq.quantity('1989/06/27/03:31:30'),'s'));
myms.select([time=[start, stop]]);
```

This example selects all the data from the measurement set where the value in the DATA_DESC_ID column is zero. This corresponds to a particular spectral window and polarization setup. It then selects all the data where the first antenna in the interferometer is number one, three or five and where the uv distance is between 1200 and 1900 meters. Finally it selects all the data which was observed between 4121629420 seconds and 4121638290 seconds (since zero hours on the day where the modified Julian day is zero). Since this time in seconds is quite obscure I have also illustrated how to use the quanta tool to convert a date/time string into seconds which can then be used to perform the same time selection.
The selections are cumulative so that at the end of this example only data in the specified time range, with the specified, interferometers, uv distances, spectral window and polarization setup are selected.

---

ms.selecttaql - Function

Package	**general**
Module	**ms**
Tool	**ms**

Select a subset of the measurement set.

Synopsis
selecttaql(msselect)

Arguments

msselect	in	TaQL selection string Allowed:String Default: no default

Returns
Boolean

Description
This function will select a subset of the current measurement set based on the standard TaQL selection string given.
Repeated use of this function, with different arguments, will further refine the selection, resulting in a successively smaller selected measurement set. If the selected measurement set does not contain any rows then this function will return F and send a warning message in the logger. Otherwise this function

will return T. To undo all the selections you need to use the selectinit function (with reset=T). Note that index values used in the TaQL string are zero-based, whereas the other commands in this tool use one-based indices.

Example

```
include 'ms.g'
myms := ms("3C273XC1.ms");
myms.selectinit(datadescid=1)
myms.select([antenna1=[1,3,5],uvdist=[1200,1900]]);
myms.selecttaql('ANTENNA1==2');
myms.range("ANTENNA1 ANTENNA2")
[antenna1=3, antenna2=[7 10 12 19 21 22 25] ]
```

This example selects all the data from the measurement set where the value in the `DATA_DESC_ID` column is zero. This corresponds to a particular spectral window and polarization setup. It then selects all the data where the first antenna in the interferometer is number one, three or five and where the uv distance is between 1200 and 1900 meters. Finally it uses a query to select all the data for which the ANTENNA1 column is 2 (this selects the middle antenna of the previous, one-based, selection). The selections are cumulative so that at the end of this example only data in the specified time range, with the specified, interferometers, uv distances, spectral window and polarization setup are selected.

---

ms.selectchannel - Function

Package	**general**
Module	**ms**
Tool	**ms**

Select and average frequency channels

Synopsis
selectchannel(nchan, start, width, inc)

Arguments

nchan	in	Number of output channels Allowed:Positive integer Default: no default
start	in	First input channel to use Allowed:Positive integer Default: no default
width	in	Number of input channels to average together Allowed:Positive integer Default: no default
inc	in	Increment to next (group of) input channel(s) Allowed:Positive integer Default: no default

Returns
Boolean

Description
This function allows you to select a subset of the frequency channels in the current measurement set. This function can also average, over frequency channels, prior to providing the values to the user.
Selection on channels is not allowed using either the select or command functions as they can only select entire rows in a measurement set. Channel selection involves accessing only some of the values in a row. Like all the selection functions this function does not change the current measurement but updates the measurement set selection parameters so that functions like getdata will return the desired subset of the data. Repeated use of this function will result overwrite any previous channel selection.
There are four parameters, the number of output channels, the first input channel to use, the number of input channels to average into one output channel, and the increment in the input spectrum for the next output channel. All four parameters need to be specified.
This function return T if the selection was successful, and F if not. In the latter case an error message will also be sent to the logger.
You need to call selectinit before calling this function. If you haven't then selectinit will be called for you with default arguments.

Example

```
include 'ms.g'
myms := ms("3C273XC1.ms");
myms.selectinit(datadescid=1);
myms.selectchannel(3,2,5,3);
```

This example selects all the data from the measurement set where the value in the DATA_DESC_ID column is zero. This corresponds to a particular spectral window and polarization setup. It then selects on frequency channels to produce 3 output channels, the first output channel is the average of channels 2,3,4,5,6 in the input, the second output channel is the average of channel 5,6,7,8,9 and the third is the average of channels 8,9,10,11,12.

---

ms.selectpolarization - Function

Package	**general**
Module	**ms**
Tool	**ms**

Select and convert of polarizations

Synopsis
selectpolarization(wantedpol)

Arguments

wantedpol	The polarizations wanted Allowed:Vector of strings Default: no default

Returns
Boolean

Description
This function allows you to select a subset of the polarizations in the current measurement set. This function can also setup conversion to different polarization representations.
You specify the polarizations using a string vector. Allowable strings are include I, Q, U, V, RR, RL, LR, LL, XX, YY, XY, YX. These string must be specified in upper case. If the polarizations match those present in the measurement set they will be selected directly, otherwise all polarizations are read and then a conversion step is done. If the conversion cannot be done then an error will be produced when you try to access the data.
This function return T if the selection was successful, and F if not.
You need to call selectinit before calling this function. If you haven't then selectinit will be called for you with default arguments.

Example

```
include 'ms.g'
myms:=ms("3C273XC1.ms");
myms.selectinit(datadescid=1);
myms.selectpolarization("I V");
rec := myms.getdata("data");
```

This example selects all the data from the measurement set where the value in the `DATA_DESC_ID` column is zero. This corresponds to a particular spectral window and polarization setup. It then selects the I and V polarizations and when the getdata function is called the conversion from RR, LL, LR, RL polarizations to I and V occurs.

---

ms.getdata - Function

Package	**general**
Module	**ms**
Tool	**ms**

Read values from the measurement set.

Synopsis
getdata(items, ifraxis, ifraxisgap, increment, average)

Arguments

items	in	Item names Allowed:Vector of strings Default: no default
ifraxis	in	Create interferometer axis if true Allowed:Boolean Default: F
ifraxisgap	in	Gap size on ifr axis when antenna1 changes Allowed:Integer Default: 0
increment	in	Row increment for data access Allowed:Integer Default: 1
average	in	Average the data in time or over rows Allowed:Boolean Default: F

Returns
record with fields containing arrays of data

Description
This function will read the specified items from the currently selected measurement set and returns them in fields of a record. The main difference between this and direct access of the table, using the table tool, is that this function reads data from the *selected* measurement set, it provides access to derived quantities like amplitude & flag_sum and it can reorder the data.
The items to read are specified, as with the range function, using a vector of strings. Table 1.355 shows the allowable items. Unrecognized items will result in a warning being sent to the logger. Duplicate items are silently ignored.
The record that is returned contains fields that correspond to each of the specified items. Most fields will contain an array. The array may be one, two or three dimensional depending on whether the corresponding row in the measurement set is a scalar, one or two dimensional. Unless the ifraxis argument is set to T the length of the last axis on these arrays will correspond to the number of rows in the selected measurement set.
If the ifraxis argument is set to T, the row axis is split into an interferometer axis and a time axis. For example a measurement set with 90 rows, in an array with 6 telescopes (so that there are 15 interferometers), may have a data array of shape `[4,32,90]` if ifraxis is F or `[4,32,15,6]`, if ifraxis is T (assuming there are 4 correlations and 32 channels). If there are missing rows as will happen if not all interferometers where used for all time-slots then a default value will be inserted.
This splitting of the row axis may not happen for items where there is only a single value per row. For some items the returned vector will contain only as many values as there are interferometers and it is implicit that the same value should be used for all time slots. The antenna1, antenna2, feed1, feed2 & ifr_number items fall in this category. For other items the returned vector will have as many values as there are time slots and it is implicit that the same value should be used for all interefometers. The field_id, scan_number, data_desc_id & time items fall into this category.
The axis_info item provides data labelling information. It returns a record with the following fields; corr_axis, freq_axis, ifr_axis & time_axis. The latter two fields are not present if ifr_axis is set to F. The corr_axis field contains a string vector with elements like 'RR' or 'XY' that indicates which polarizations where correlated together to produce the data. The length of this vector will always be the same as the length of the first axis of the data array. The freq_axis field contains a record with two fields, chan_freq & resolution. Each of these fields contains vectors which indicate the centre frequency and spectral resolution (FWHM) of each channel. The length of these vectors will be the same as the length of the second axis in the data. The ifr_axis field contains for fields; ifr_number, ifr_name, ifr_shortname & baseline. The ifr_number is the same as returned by the ifr_item, the ifr_name

& ifr_shortname are string vecors containing descriptions of the interferometer and the baseline is the Euclidian distance, in meters between the two antennas. All of these vectors have a length equal to the number of interferometers in the selected measurement set ie., to the length of the third axis in the data when ifraxis is T. The time_axis field contains the MJDseconds field and optionally the HA, UT & LAST fields. To include the optional fields you need to add the ha, last or ut strings to the list of requested items. All the fields in the time_axis record contain vectors that indicate the time at the midpoint of the observation and are in seconds. The MJDseconds is since 0 hours on the day having a modified julian day number of zero and the rest are since midnight prior to the start of the observation.
An optional gap size can be specified to visually separate groups of interferometers with the same antenna1 index (handy for identifying antennas in an interferometer vs time display). The default is no gap.
An optional increment can be specified to return data from every row matching the increment only.
When the average flag is set, the data will be averaged over the time axis if the ifraxis is T or the row axis i.e., different interferometers and times may be averaged together. In the latter case, some of the coordinate information, like antenna_id, will no longer make sense.
You need to call selectinit before calling this function. If you haven't then selectinit will be called for you with default arguments.
Items prefixed with either; corrected, model, residual or obs_residual and the imaging_weight item are not available unless your measurement set has been processed either with the imager or calibrator tools.

Example

```
include 'ms.g'
myms := ms("3C273XC1.ms")
myms.selectinit(datadescid=1)
d := myms.getdata("amplitude ifr_axis ha")
include 'pgplotter.g'
pg := pgplotter()
tstart := min(z.axis_info.time_axis.HA)
tstop := max(z.axis_info.time_axis.HA)
maxamp := max(z.amplitude)
pg.env(tstart, tstop, 0,  maxamp, 0, 1)
corr := 1;
ifr := 1;
chan := 1;
pg.pt(d.axis_info.time_axis.HA, d.amplitude[corr, ifr, chan,,], 2)
title := paste('Frequency',
              as_string(d.axis_info.freq_axis.chan_freq[chan]/1E9),'GHz',
```

```
                  'Baseline', d.axis_info.ifr_axis.ifr_name[corr],
                  '(', d.axis_info.corr_axis[corr], ')')
      pg.lab('Hour angle (sec)', 'Correlation amplitude', title)
```

This example selects all the data from the measurement set where the value in the `DATA_DESC_ID` column is zero. This corresponds to a particular spectral window and polarization setup. It then gets the correlated amplitude, and the axis information from this selected measurement set. This is returned in the *Glish* variable d. The remainder of the example uses the pgplotter tool to make a plot of correlated amplitude vs hour angle for the first channel, correlation and baseline.

---

ms.putdata - Function

Package	**general**
Module	**ms**
Tool	**ms**

Write new values into the measurement set

Synopsis
putdata(items)

Arguments

items	in	Record with items and their new values Allowed:Record Default: no default

Returns
Boolean

Description
This function allows you to write values from *Glish* variables back into the measurement set table. The main difference between this and direct access of the table, using the table tool, is that this function writes data to the *selected* measurement set.
Unlike the getdata function you can only put items that correspond to actual table columns. You cannot change the data shape either so that the number of correlations, channels and rows (or intereferometers/time slots) must match

the values in the selected measurement set. If the values were obtained using the getdata function with ifraxis argument set to T, then,any default values added to fill in missing intereferometer/timeslots pairs will be ignored when writing the modified values back using this function.
The measurement set has to be opened for read/write access to be able to use this function.
You need to call selectinit before calling this function. If you haven't then selectinit will be called for you with default arguments.
Items prefixed with either; corrected, model, residual or obs_residual and the imaging_weight item are not available unless your measurement set has been processed either with the imager or calibrator tools.

Example

```
include 'ms.g'
myms := ms("3C273XC1.ms", readonly=F)
myms.selectinit(datadescid=1)
rec := myms.getdata("weight data")
rec.weight[,] := 1;
include 'statistics.g'
rec.corrected_data -:= mean(rec.corrected_data);
myms.putdata(rec);
```

This example selects all the data from the measurement set where the value in the `DATA_DESC_ID` column is zero. This corresponds to a particular spectral window and polarization setup. Note that the measurement set was opened for writing as well as reading. The fourth line reads all the weights and the corrected data into the *Glish* variable rec. The weights are set to one. The more obscure syntax is used as typing `rec.weight:= 1` will not preserve the shape of the weight array. The corrected data then has its mean subtracted from it. The mean function is defined in statistics.g. Finally the data is written back into the measurement set table.

---

ms.concatenate - Function

Package	**general**
Module	**ms**
Tool	**ms**

Concatenate two measurement sets

Synopsis
concatenate(msfile, freqtol, dirtol)

Arguments

msfile	in	The name of the measurement set to append Allowed:String Default: no default
freqtol	in	Frequency difference within which 2 spectral windows are considered similar Allowed:Quantity Default: '1Hz'
dirtol	in	Direction difference within which 2 fields are considered the same Allowed:Quantity Default: '1mas'

Returns
Nothing

Description
This function concatenates two measurement sets together.
The data is copied from the measurement set specified in the msfile arguement to the end of the measurement set attached to the ms tool. If a lot of data needs to be copied this function may take some time. You need to open the measurement set for writing in order to use this function.
This function is new to the ms tool and currently requires that both the measurement sets involved in the concatenation were originally derived from of a UVFITS file (and converted using the fitstoms function).

Example

```
include 'ms.g'
myms := ms("3C273XC1.ms", readonly=F)
myms.concatenate("BLLAC.ms", '1GHz', '1arcsec')
myms.done()
```

This example appends the data from the BLLAC measurement set to the end of the 3C273 measurement set. Its going to assume a frequency tolerance of 1GHz and position tolerance of 1 arcsec in deciding if the spwids and fieldids in the measurementsets are similar or not.

---

ms.split - Function

Package	**general**
Module	**ms**
Tool	**ms**

make a new ms from a subset of actual ms

Synopsis
split(outputms, fieldids, spwids, nchan, start, step, timebin, timerange, whichcol)

Arguments

outputms	in	The name of the measurement split into Allowed:String Default: no default
fieldids	in	Field ids to split out 1-based Allowed:Vector Int Default: -1 (i.e all fields)
spwids	in	Spectral windows to split 1-based Allowed:Vector Int Default: -1 (i.e all spws)
nchan	in	number of channels in output Allowed:Vector Int, length 1 or same length as spwids Default: -1 (i.e all channels in spws)
start	in	Start channels in input data Allowed:Vector Int, length 1 or same length as spwids Default: 1
step	in	number of input channels to average to make 1 output channel Allowed:Vector Int, length 1 or same as spwids Default: 1
timebin	in	Value for time averaging Allowed:Quantity Default: '-1s'
timerange	in	Selection of time range to split out; MSSelection syntax Allowed:String Default: "
whichcol	in	'DATA', 'MODEL_DATA', 'CORRECTED_DATA' Allowed:String Default: 'DATA'

Returns
Nothing

Description
This function split out part of the MS into a new one. Spectral averaging is available for now on splitting. More transformations are in the works (like time averaging).
When splitting multiple spectral windows, the parameters `nchan, start, step` can be vectors, so that each spectral window has its own selection on averaging and number of output channels. But the option of using only one value for each of these parameters means that it will be replicated for all the spectral windows selected.
If time averaging is needed the `timebin` parameter should be set to the requested time integration the visibilities should be in. If `timebin` is set to a value which is smaller (or just less than 0) than the minimum integration time in the input ms then no time averaging will be done while splitting. Please note that if there are spectral windows of different shapes in the selection to be splitted out then time averaging is not available yet. The way around is to split each spectral window with time averaging seperately into different ms's and then concatenate them together afterwards.
The parameter `timerange` allows for data selection over time. The syntax is defined in the msselection syntax document; the relevant section is quoted here:

```
timerange = 'YYYY/MM/DD/HH:MM:SS.sss'
     = '< YYYY/MM/DD/HH:MM:SS.sss'
     = '> YYYY/MM/DD/HH:MM:SS.sss'
     = 'ddd/HH:MM:SS.sss'
     = '< ddd/HH:MM:SS.sss'
     = '> ddd/HH:MM:SS.sss'

Examples:
timernage = '2003/11/07/12:58:20'        # selects the timestamp nearest this time

timerange = '2003/11/07/12:58:20-45'     # selects data within this 25s range

timerange = '2003/11/07/12:58:20-59:45'  # selects data within this 1m25s range

timerange = '< 2003/11/07/12:58:20'     # selects data prior to this time

timerange = '13:05:10.005'              # selects timestamp nearest this time
                                   (date defaults to first date in dataset)
timerange = '0/13:05:10.005'            # same as above

timerange = '3/13:05:10.005'            # selects timestamp nearest this time on 4th day in dataset
```

```
timerange = '13:05'                     # selects timestamp nearest 13h05m (date defaults to first date in  dataset)

timerange = '< 13:05:10, > 13:06:35'    # all but the 1m25s of data between                                   these times (date defaults to first date in data)
```

Example

```
      include 'ms.g'
      myms := ms("multiwin.ms", readonly=F)
      myms.split('subms.ms', fieldids=[1], spwids=[1], nchan=[10],
  start=[1], step=[5], whichcol='CORRECTED_DATA')
```

In this example we split out data from the 1st field and 1st spectral window. The output data will have 10 channels which is taken from 50 channels from the input data start at channel 1 and averaging every 5.

```
      include 'ms.g'
      myms := ms("multiwin.ms", readonly=F)
      myms.split('subms.ms', fieldids=[1], spwids=[1,2,3,4], nchan=[10],
  start=[1], step=[5], whichcol='CORRECTED_DATA')
```

In this example we split out data from the 1st field and four spectral windows. spectral window. The output data will have 4 spectral windows each of 10 channels which is taken from 50 channels from the input data start at channel 1 and averaging every 5.

```
      include 'ms.g'
      myms := ms("multiwin.ms", readonly=F)
      myms.split('subms.ms', fieldids=[1], spwids=[1,2,3,4], nchan=[10,10,30,40],
  start=[1,5,10,10], step=[1,10,5,2], whichcol='CORRECTED_DATA')
```

In this example we split out data from the 1st field and four spectral windows. There will be four spectral windows in the output data, with 10, 20, 30 and 40 channels respectively. These are averages of the input spectral windows. The first output spectral window will be formed by picking 10 channels, starting at 1 with no averaging, of the input spwid 1. The second output spectral window will consists of 10 channels and is formed by picking 100 channels from spwid 2 of the input data, starting at channel 5, and every 10 channels to make one output channel.

---

ms.iterinit - Function

Package	**general**
Module	**ms**
Tool	**ms**

Initialize for iteration over a MS

Synopsis

iterinit(columns, interval, maxrows, adddefaultsortcolumns)

Arguments

columns	in	Double quoted string with column names Allowed:Vector of strings Default: no default
interval	in	Time interval in seconds, to group together in iteration Allowed:Positive number Default: no default
maxrows	in	Max number of rows to return in iteration Allowed:Positive integer Default: no default
adddefaultsortcolumns	in	Add the default sort columns Allowed:Boolean Default: T

Returns

Boolean

Description

Specify the columns to iterate over and the time interval to use for the TIME column iteration. The columns are specified by their MS column name. Note that the following columns are always added to the specified columns: array_id, field_id, data_desc_id and time. This is so that the iterator can keep track of the coordinates associated with the data (field direction, frequency etc.) If you want to sort on these columns last instead of first you need to include them in the columns specified. If you don't want to sort on these columns at all, you can set adddefaultsortcolumns to F. You need to call selectinit before calling this. See the example below.

Example

See the example for the iterend function.

ms.iterorigin - Function

Package	**general**
Module	**ms**
Tool	**ms**

Set the iterator to the start of the data.

Synopsis
iterorigin()

Returns
Boolean, True if operation successful

Description
Set or reset the iterator to the start of the currently specified iteration, you need to call this before attempting to retrieve data with getdata. You can set the iteration back to the start before you reach the end of the data. You need to call iterinit before calling this. See the example below.

Example
See the example for the iterend function.

ms.iternext - Function

Package	**general**
Module	**ms**
Tool	**ms**

Advance the iterator to the next lot of data

Synopsis
iternext()

Returns
Bool, True if there was more data, False if we've reached the end

Description
This sets the currently selected table (as accessed with getdata) to the next iteration. If there is no more data, the function returns False and the selection is reset to that before the iteration started. You need to call iterinit and iterorigin before calling this. See the example below.

Example
See the example for the iterend function.

---

ms.iterend - Function

Package	**general**
Module	**ms**
Tool	**ms**

End the iteration and reset the selected table

Synopsis
iterend()

Returns
Bool, True if iteration was in progress, False if not

Description
This sets the currently selected table (as accessed with getdata) to the table that was selected before iteration started. Use this to end the iteration prematurely. There is no need to call this if you continue iterating until iternext returns False.
See the example below.

Example

```
include 'ms.g'
myms:=ms("3C273XC1.MS");
myms.selectinit(datadescid=1)
myms.iterinit("antenna1 antenna2 time",60.0);
myms.iterorigin();
rec:=myms.getdata("u v data");
myms.iternext();
```

```
    myms.iterend();
```

We open the MS, select an array and spectral window and then specify an iteration over interferometer and time, with a 60s time interval. We then set the iterator to the start of the data and get out some data. Finally we advance the iterator to the next lot of data and then end the iteration.

---

ms.tosdfits - Function

Package	**general**
Module	**ms**
Tool	**ms**

Convert a single dish AIPS++ measurement set to a SDFITS file

Synopsis
tosdfits(fitsfile)

Arguments

fitsfile	Name of the new SDFITS file Allowed:String

Description

This function turns an MS tool into a FITS file containing an SDFITS binary table. This uses the `ms2sdfits` program to do the conversion. It converts the entire underlying measurement set file even if the ms tool is the result of a selection. This may be changed in the future. The ms tool must consist of single dish data only. A brief description of SDFITS may be found in the **dish** users manual. Converting a measurement set into an SDFITS binary table will generally result in the loss of some information which can not be represented in an SDFITS binary table.

Example

```
- include 'ms.g'
- myms := ms('myfile.ms')         # open the MS
- myms.tosdfits("myfile.sdfits")  # write SDFITS file
```

---

ms.createflaghistory - Function

Package	**general**
Module	**ms**
Tool	**ms**

Create and initialize the FLAG_HISTORY column.

Synopsis
createflaghistory(numlevel)

Arguments

numlevel	Number of history levels Allowed:Positive integer Default: 2

Returns
Bool, True if operation successful

Description

Create the FLAG_HISTORY column and fill level 1 and 2 with the current flagging state as given by FLAG and FLAG_ROW. Then sets the current flagging level to 2.

Example

```
myms:=ms("3C273XC1.MS",readonly=F);
myms.createflaghistory(3);
```

We open the MS for writing and then create a FLAG_HISTORY column with 3 levels of flagging.

---

ms.saveflags - Function

Package	**general**
Module	**ms**
Tool	**ms**

Save the current flagging state to the FLAG_HISTORY column.

Synopsis
saveflags(newlevel)

Arguments

newlevel	Copy to a new level? Allowed:Bool Default: F

Returns
Bool, True if operation successful

Description
Copies the flags in the FLAG and FLAG_ROW, to the FLAG_HISTORY column. Copies to the current level by default and to a new level, if available, when newlevel=T.

Example

```
myms:=ms("3C273XC1.MS",readonly=F);
myms.saveflags();
```

We open the MS for writing and then copy the flags to the current level (see flaglevel() function) in the FLAG_HISTORY column, overwriting the previous values.

---

ms.restoreflags - Function

Package	**general**
Module	**ms**
Tool	**ms**

Restore the specified flagging level from the FLAG_HISTORY column.

Synopsis
restoreflags(level)

Arguments

level	History level to copy from Allowed:Int Default: 0 = current level

Returns
Bool, True if operation successful

Description
Copies the flags in the FLAG_HISTORY column (for the specified level) back to the FLAG FLAG_ROW column. See the example below.

Example

```
myms:=ms("3C273XC1.MS",readonly=F);
myms.restoreflags(2);
```

We open the MS for writing and then copy the flags from level 2 in the FLAG_HISTORY column to the FLAG and FLAG_ROW column, overwriting the previous values.

---

ms.flaglevel - Function

Package	**general**
Module	**ms**
Tool	**ms**

return the current flagging level

Synopsis
flaglevel()

Returns
Int, the current flag level, or 0 if none exists

Description
Return the currently active flagging level of the FLAG_HISTORY column as specified by the FLAG_LEVEL keyword of this column.

Example

```
myms:=ms("3C273XC1.MS");
flaglev:=myms.flaglevel();
```

We open the MS and ask for the current flag level.

---

ms.fillbuffer - Function

Package	**general**
Module	**ms**
Tool	**ms**

Fill the internal buffer with data and flags.

Synopsis

fillbuffer(item, ifraxis)

Arguments

item	data derived item Allowed:String
ifraxis	Create interferometer axis if true Allowed:Bool Default: F

Returns

Bool, True if operation successful

Description

Read the specified data item from the table, including its flags and keep the results in an internal buffer

Example

```
myms:=ms("3C273XC1.MS");
myms.select([antenna1=3]);
myms.fillbuffer("PHASE",T);
```

We open the MS for reading, select a subset and then read the DATA, FLAG and FLAG_ROW column, extract the PHASE, reorder the data to add an interferometer axis, and keep the results around in an internal buffer.

---

ms.diffbuffer - Function

Package	**general**
Module	**ms**
Tool	**ms**

Differentiate or difference the internal buffer.

Synopsis
diffbuffer(direction, window)

Arguments

direction	choose between time or channel direction Allowed:String: TIME or CHANNEL
window	width of averaging window in timeslots or channels Allowed:Positive integer

Returns
Record, with statistics over the buffer if all went well

Description
Subtract the previous point from each data point in the buffer (for window=2), or subtract the average over a window (for window¿2) from each point. The window can be in the time / row direction or the frequency / channel direction. The input data can be float or complex but the output is always float. The function returns statistics over the buffer: median for each time and channel, the average absolute deviation from the median in time and channel direction and over all pixels.

Example

```
myms:=ms("3C273XC1.MS");
myms.select([antenna1=3]);
myms.fillbuffer("DATA");
myms.diffbuffer("TIME",15)
```

We open the MS for reading, select a subset and then read the DATA, FLAG and FLAG_ROW column, we then subtract the average over a 15 point time-window from each data point.

---

ms.getbuffer - Function

Package	**general**
Module	**ms**
Tool	**ms**

Return the internal buffer as a Record for access from Glish.

Synopsis
getbuffer()

Returns
Record with data if operation successful

Description
Returns the internal buffer with either 'raw' or differenced data, flags and buffer statistics (if a difference operation was performed).

Example

```
myms:=ms("3C273XC1.MS");
myms.select([antenna1=3]);
myms.fillbuffer("PHASE");
rec:=myms.getbuffer();
```

We open the MS for reading, select a subset and then read the DATA, FLAG and FLAG_ROW column, extract the PHASE and then obtain the results in a record.

---

ms.clipbuffer - Function

Package	**general**
Module	**ms**
Tool	**ms**

Clip the internal buffer with specified limits.

Synopsis
clipbuffer(pixellevel, timelevel, channellevel)

Arguments

pixellevel	cliplevel for pixels Allowed:positive value
timelevel	cliplevel for time slots Allowed:positive value
channellevel	cliplevel for channels Allowed:positive value

Returns
Bool, True if operation successful

Description
This sets flags in the internal buffer based on the clip levels specified. You can flag times, channels and individual pixels based on their deviation from the median. The cliplevel is specified in units the corresponding average absolute deviation (a robust version of rms).

Example

```
mymms:=ms("3C273XC1.MS");
mymms.select([antenna1=3]);
mymms.fillbuffer("DATA");
stats:=mymms.diffbuffer("TIME",15);
mymms.clipbuffer(6,5,5);
```

We open the MS for reading, select a subset and read the data into the buffer. We then remove the average over a 15 point time window and clip the resulting data at 6 times the average absolute deviation from the median for individual pixels, and at 5 times this for channels and timeslots.

---

ms.setbufferflags - Function

Package	**general**
Module	**ms**
Tool	**ms**

Set the flags in the buffer

Synopsis
setbufferflags(flags)

Arguments

flags	record with flag and flag_row Allowed:Record	

Returns
Bool, True if operation successful

Description
Replace the flag and flag_row fields in the internal buffer with those in the input record. The input record can e.g., be a modified version of the record returned by getbuffer(). The other fields in the record are ignored.

Example

```
myms:=ms("3C273XC1.MS",F);
myms.select([antenna1=3]);
myms.fillbuffer("PHASE");
rec:=myms.getbuffer();
rec.flag_row[17]:=T;
myms.setbufferflags(rec);
```

We open the MS for reading, select a subset and read the data. We get the data into Glish, flag timeslot 17 and put the modified flags back into the buffer.

---

ms.writebufferflags - Function

Package	**general**
Module	**ms**
Tool	**ms**

Write the flags in the internal buffer back to the table.

Synopsis
writebufferflags()

Returns
Bool, True if operation successful

Description
Takes the flag and flag_row field in the internal buffer and writes them back to the FLAG and FLAG_ROW column in the currently selected table.

Example

```
myms:=ms("3C273XC1.MS",F);
myms.select([antenna1=3]);
myms.fillbuffer("PHASE");
rec:=myms.getbuffer();
rec.flag_row[17]:=T;
myms.setbufferflags(rec);
myms.writebufferflags();
```

We open the MS for reading, select a subset and read the data. We get the data into Glish, flag timeslot 17 and put the modified flags back into the buffer. We then write the buffer flags back to the table, causing the corresponding data to be marked flagged on subsequent access.

---

ms.clearbuffer - Function

Package	**general**
Module	**ms**
Tool	**ms**

Clear the internal buffer.

Synopsis
clearbuffer()

Returns
Bool, True if operation successful

Description
Clears the internal buffer, returning the memory. This can be used after the (final) clipping/flagging operations have been performed.

Example

```
myms:=ms("3C273XC1.MS",F);
myms.select([antenna1=3]);
myms.fillbuffer("PHASE");
rec:=myms.getbuffer();
rec.flag_row[17]:=T;
myms.setbufferflags(rec);
myms.writebufferflags();
myms.clearbuffer();
```

We open the MS for reading, select a subset and read the data. We get the data into Glish, flag timeslot 17 and put the modified flags back into the buffer. We then write the buffer flags back to the table, causing the corresponding data to be marked flagged on subsequent access. Finally we clear the internal buffer. This step can be omitted if you are about to do another fillbuffer().

---

ms.continuumsub - Function

Package	**general**
Module	**ms**
Tool	**ms**

Continuum fitting and subtraction in uv plane

Synopsis
continuumsub(fldid, spwid, chans, solint, fitorder, mode)

Arguments

fldid	in	Desired fields to fit Allowed:Vector of integers Default: no default
spwid	in	Desired spectral windows to fit Allowed:Vector of integers Default: all
chans	in	Range of channels to fit Allowed:Vector of integers Default: all
solint	in	Averaging time (s) Allowed:Float Default: 0.0 (scans)
fitorder	in	Polynomial order for fit Allowed:Integer Default: 0
mode	in	Desired use of fit model (see below) Allowed:String Default: 'subtract'

Description

This function does exactly the same as uvlsf, and will replace that function in future when it proves to be sufficiently bug free. Continuumsub is written in C++ as opposed to glish and is faster by a factor 3-10 (depends on size of dataset and solution interval). This function provides a means of continuum determination and subtraction by fitting a polynomial of desired order to a subset of channels in each time-averaged uv spectrum. The fit is used to model the continuum in all channels (not just those used in the fit), for subtraction, if desired. Use the `chans` parameter to limit the range of channels used to estimate the continuum (avoid channels containing spectral lines). The default solution interval (0.0s) will average the data over scans (per-baseline) before doing the fit. The `mode` parameter indicates how the continuum model (the result of the fit) should be used: `'subtract'` will store the continuum model in the MODEL_DATA column and subtract it from the CORRECTED_DATA column; `'replace'` will replace the CORRECTED_DATA column with the continuum model (useful if you want to image the continuum model result); and `'model'` will only store the continuum model in the MODEL_DATA column (the CORRECTED_DATA is unaffected).

It is important to start the ms tool with `readonly=F` so that changes to the dataset will be allowed (see example below).

For now, the only way to recover the un-subtracted CORRECTED_DATA column is to use calibrater.correct() again.

Note that the MODEL_DATA and CORRECTED_DATA columns must be present for `continuumsub` to work correctly. The function will warn the user if they are not present, and abort. To add these scratch columns (for now), done the ms tool, then start a calibrater or an imager tool, which will add the scratch columns. Then restart the ms tool, and try `continuumsub` again. Options for shifting known bright sources to the phase center and for editing based on the rms fit will be added in the near future.

Example

```
include 'ms.g'
myms := ms('ngc5921.ms',readonly=F);    # writable!
myms.continuumsub(fldid=3,spwid=1,chans=[5:9,50:59],
           solint=0.0,fitorder=1,mode='sub');
myms.done();
```

This example will fit a line to channels 5-9 and 50-59 in spectral window 1 in each scan-averaged spectrum for field 3, and store the result in the MODEL_DATA column and subtract it from the CORRECTED_DATA column.

---

ms.uvlsf - Function

Package	**general**
Module	**ms**
Tool	**ms**

Continuum fitting and subtraction in uv plane

Synopsis
uvlsf(fldid, spwid, chans, solint, fitorder, mode)

Arguments

fldid	in	Desired fields to fit Allowed:Vector of integers Default: no default
spwid	in	Desired spectral windows to fit Allowed:Vector of integers Default: all
chans	in	Range of channels to fit Allowed:Vector of integers Default: all
solint	in	Averaging time (s) Allowed:Float Default: 0.0 (scans)
fitorder	in	Polynomial order for fit Allowed:Integer Default: 0
mode	in	Desired use of fit model (see below) Allowed:String Default: 'subtract'

Description

This function provides a means of continuum determination and subtraction by fitting a polynomial of desired order to a subset of channels in each time-averaged uv spectrum. The fit is used to model the continuum in all channels (not just those used in the fit), for subtraction, if desired. Use the `chans` parameter to limit the range of channels used to estimate the continuum (avoid channels containing spectral lines). The default solution interval (0.0s) will average the data over scans (per-baseline) before doing the fit. The `mode` parameter indicates how the continuum model (the result of the fit) should be used: `'subtract'` will store the continuum model in the MODEL_DATA column and subtract it from the CORRECTED_DATA column; `'replace'` will replace the CORRECTED_DATA column with the continuum model (useful if you want to image the continuum model result); and `'model'` will only store the continuum model in the MODEL_DATA column (the CORRECTED_DATA is unaffected).

It is important to start the ms tool with `readonly=F` so that changes to the dataset will be allowed (see example below).

For now, the only way to recover the un-subtracted CORRECTED_DATA column is to use calibrater.correct() again.

Note that the MODEL_DATA and CORRECTED_DATA columns must be present for `uvlsf` to work correctly. The function will warn the user if they are not present, and abort. To add these scratch columns (for now), done the ms tool, then start a calibrater or an imager tool, which will add the scratch columns. Then restart the ms tool, and try `uvlsf` again.

Options for shifting known bright sources to the phase center and for editing based on the rms fit will be added in the near future.

Example

```
include 'ms.g'
myms := ms('ngc5921.ms',readonly=F);    # writable!
myms.uvlsf(fldid=3,spwid=1,chans=[5:9,50:59],
           solint=0.0,fitorder=1,mode='sub');
myms.done();
```

This example will fit a line to channels 5-9 and 50-59 in spectral window 1 in each scan-averaged spectrum for field 3, and store the result in the MODEL_DATA column and subtract it from the CORRECTED_DATA column.

---

ms.ptsrc - Function

Package	**general**
Module	**ms**
Tool	**ms**

Reports global average of visibilities per field

Synopsis
ptsrc(fldid, spwid)

Arguments

fldid	in	Desired fields Allowed:Vector of integers Default: all
spwid	in	Desired spectral windows Allowed:Vector of integers Default: all

Description
This function averages the data over all (unflagged) times and (non-zero) baselines per field_id and spectral_window_id, and reports the result in the Stokes polarization basis. If present, the CORRECTED_DATA column is averaged; otherwise the DATA column is used, in which case the results might be suspect if this DATA hasn't been calibrated.
For multi-channel data, all channels are currently averaged as well. Future versions of this function will have options for channel selection.

This function is useful for obtaining a full-Stokes point source flux density estimate for polarization calibration or other purposes. It is essentially calculating a single-pixel image at the observation's phase center, which for point sources, yields the correct flux density.

Example

```
include 'ms.g'
myms := ms('ap366.ms');
myms.ptsrc(fldid=[9,11],spwid=1);
myms.done();
```

This example calculates the point source flux density for field ids 9 & 11 in spectral window 1.

---

ms.done - Function

Package	**general**
Module	**ms**
Tool	**ms**

Delete the ms `tool`

Synopsis
done()

Description
You should call this function when you are finished using the ms `tool`. It frees up all the memory associated with a ms tool, closes the measurement set table and frees up any associated file locks. The measurement set is not deleted. It will also shut down the server process if there are no other ms tools being used.

Example

```
myms := ms("3C273XC1.ms");
...
myms.done();
```

1.3.2 is_ms - Function

Package general
Module ms

Is the argument a ms tool?

Synopsis
is_ms(tool)

Description
This global function can be used to determine if the supplied argument is a ms tool. If so it returns True, otherwise it returns False.

Arguments

tool	The variable that you wish to test Allowed: Any glish variable Default: no default

Returns
T or F, indicating if the argument is a componentlist tool or not.

Example

```
include 'ms.g';
myms := ms('3C273XC1.ms');
if (is_ms(myms)) {
  print 'The ms name is', myms.name();
}
```

This example shows how you can use this function to determine if the supplied variable is a ms tool.

1.3.3 msfiles - Function

Package general
Module ms